Universidad de **Valladolid**

Departamento de Informática

Estudio y Aplicación de Nuevos Métodos de Compresión de Texto Orientada a Palabras.

– Tesis Doctoral –

Doctorado en Informática

Miguel Ángel Martínez Prieto

Directores de la tesis:

Dr. Joaquín Adiego Rodríguez
Departamento de Informática
Universidad de Valladolid
Valladolid, España

Dr. Pablo de la Fuente Redondo
Departamento de Informática
Universidad de Valladolid
Valladolid, España

* Este trabajo ha sido desarrollado al amparo de una ayuda para la Formación de Personal Investigador financiada por la Junta de Castilla y León -Consejería de Educación- y el Fondo Social Europeo -FSE-.

** Parte de los resultados contenidos en este trabajo han sido obtenidos bajo los objetivos indicados en el proyecto TIN2009-14009-C02-02 concedido por el Ministerio de Ciencia e Innovación (España).

Resumen

La demanda de información se ha multiplicado en los últimos años gracias, principalmente, a la globalización en el acceso a la WWW. Esto ha propiciado un aumento sustancial en el tamaño de las colecciones de texto disponibles en formato electrónico, cuya *compresión* no sólo permite obtener un ahorro espacial sino que, a su vez, aumenta la eficiencia de sus procesos de entrada/salida y de transmisión en red.

La *compresión de texto* trata con información expresada en lenguaje natural. Por lo tanto, la identificación de la redundancia subyacente a este tipo de textos requiere adoptar una perspectiva *orientada a palabras*, considerando ésta como la unidad mínima de información utilizada en los procesos de comunicación entre personas. Esta tesis aborda el estudio del contexto anterior desde tres perspectivas complementarias cuyos resultados se traducen en la obtención de un conjunto de compresores de texto específicos.

El lenguaje natural posee unas propiedades particulares, tanto en lo relativo al tamaño del vocabulario de palabras identificado en el texto como a la distribución de frecuencia que muestra cada una de ellas. Sin embargo, las técnicas universales de compresión no son capaces de identificar, específicamente, estas propiedades al no restringir el tipo de mensajes que toman como entrada. La primera propuesta de esta tesis se centra en la construcción de un esquema de preprocesamiento (denominado *Word-Codeword Improved Mapping*: `WCIM`) que transforma el texto original en una representación más redundante del mismo que favorece su compresión con técnicas clásicas. A pesar de su sencillez y efectividad, esta propuesta no gestiona un aspecto relevante en lenguaje natural: la relación existente entre las palabras.

La familia de técnicas *Edge-Guided* (`E-G`) utilizan la relación de adyacencia entre símbolos como base para la representación del texto. El compresor $\texttt{E-G}_1$ construye un modelo de orden 1 orientado a palabras, cuya representación se materializa en las aristas de un grafo dirigido. Por su parte, $\texttt{E-G}_k$ considera la extensión del vocabulario original con un conjunto de secuencias de palabras (frases) significativas que se representan a través de una gramática libre de contexto. El modelo de grafo original evoluciona de tal forma que pasa a representar un modelo de orden 1 orientado a frases en el que la relación de jerarquía, existente entre las palabras que las constituyen, puede ser aprovechada a través de la información almacenada en la gramática. Tanto $\texttt{E-G}_1$ como $\texttt{E-G}_k$ utilizan la información almacenada en las aristas del grafo para la construcción de sus esquema de codificación basado en un código de Huffman.

Los corpus paralelos bilingües (*bitextos*) están formados por dos textos, en lenguaje natural, que expresan la misma información en dos idiomas diferentes. Esta propiedad suma un tipo de redundancia no tratada en los casos anteriores: la redundancia *semántica*. Nuestras propuestas, en este contexto, se centran en la representación de bitextos alineados, cuya utilización es un aspecto esencial en numerosas aplicaciones relacionadas con la traducción. Para ello introducimos el concepto de *bipalabra* como unidad simbólica de representación y se plantean sendas técnicas basadas en sus propiedades estructurales (*Translation Relationship-based Compressor*: `TRC`) y semánticas (*Two-Level Compressor for Aligned Bitexts*: `2LCAB`). Ambas propuestas analizan el efecto, en la compresión, asociado al hecho de utilizar diferentes estrategias de alineamiento del bitexto. Complementariamente, `2LCAB` plantea un mecanismo de búsqueda, basado en *pattern-matching*, que permite llevar a cabo diferentes tipos de operaciones sobre el texto comprimido.

Los procesos de experimentación, llevados a cabo sobre corpus de referencia en cada uno de los contextos, demuestran la competitividad de cada una de los compresores propuestos. Los resultados obtenidos con la técnica `2LCAB` son especialmente significativos ya que soportan la primera propuesta conocida que facilita la consulta monolingüe y translingüe sobre un bitexto comprimido. Esta propiedad aísla el idioma en el que se recuperan los resultados del utilizado en la consulta, planteando `2LCAB` como una alternativa competitiva para su uso como "motor de búsqueda" en diferentes herramientas de traducción.

Agradecimientos

Que la fuerza os acompañe,
buena suerte y recordad:
"que sin vosotros
no hay cantante
ni canción".

Carlos Goñi

Es difícil describir con palabras todos aquellos motivos y personas a las que tendría que dar las gracias en este momento. Aún así, intentaré ser capaz de dejar una breve reseña de todos ellos sin dejar de recordar que todo esto *lo llevo dentro*.

Quiero agradecer a mis directores de tesis, Joaquín y Pablo, la posibilidad de descubrir este mundo de los algoritmos y las estructuras de datos aplicadas a la compresión. Haber encontrado un tema de investigación que me atrajese, como lo hace éste, es lo que ha permitido que esta tesis esté hoy finalizada. Siento no haber sido capaz de aprender mucho más, pero ahora tengo todas las ganas del mundo de hacerlo y eso es muy importante para mí.

Hay una persona imprescindible a la hora de valorar mi vida informática y aunque este es mi primer trabajo académico del cual él no es responsable, Carlos fue, es y será siempre mi *maestro*, a la vez que uno de mis grandes amigos. Gracias por todo el tiempo que invertiste en escucharme, en enseñarme y en comprenderme más allá de nuestras batallas con el XML, gracias por ayudarme a ser una mejor persona y por hacerme entender que sin eso da igual lo buen informático que pueda llegar a ser. Gracias por haber confiado en mí cuando ni yo mismo lo hacía, de no haber sido por eso estoy seguro de que este día no habría llegado nunca.

Debería aumentar el tamaño de la letra para ser capaz de reflejar mi infinito agradecimiento a Gonzalo Navarro. Conocer parte de su trabajo motivó mis ganas de seguir haciéndolo y conocerlo personalmente despertó en mí una profunda admiración que culminó (y a la vez re-empezó) en el pequeño sueño de poder trabar y aprender con él y de él. Pero mis gracias hacia Gonzalo también tienen un componente personal muy importante ya que todo el tiempo que pasé en Chile me permitió redescubrirme como persona y ganar una confianza que nunca antes tuve... y eso es algo que trasciende más allá del mundo binario.

No quiero que sea menor el agradecimiento hacia todas las personas con las que he tenido la oportunidad de trabajar directamente en todos estos años y de las que he tenido la suerte de aprender. Quiero agradecer a Nieves Brisaboa toda la paciencia y el tiempo que ha invertido conmigo en cada una de las oportunidades que hemos tenido de dialogar y trabajar en este último año. Por el mismo motivo quiero darle las gracias a Antonio Fariña, sin obviar que sus consejos y ayuda me han permitido mejorar algunas de las ideas y resultados que componen esta tesis. Quiero agradecerle a ambos, conjuntamente, y a todas las personas del *Laboratorio de Bases de Datos* el trato que me dieron los días que pasé en Coruña en los que me sentí, en todo momento, como en casa. Gracias a Felipe Sánchez por todo el trabajo que hemos compartido en estos años y que me ha permitido acercarme a un conjunto de ideas interesantes que han dado lugar a un capítulo completo de esta tesis, en el que su ayuda también ha sido importante. Gracias a tí y a todas las personas del grupo *Transducens* que he conocido porque cada visita a Alicante hace que aumenten mis ganas de regresar. Gracias a Diego Arroyuelo, Yigo Cánovas, Francisco Claude y a Susana Ladra por haberme ayudado a entender y utilizar sus estructuras de datos y algoritmos, sin vosotros me habría resultado muy difícil. Y gracias también a todas las personas que forman el grupo *Miércoles de Algoritmos* en la *Universidad de Chile* por hacerme sentir uno más entre todos ellos y por sus sugerencias y comentarios en cada una de las presentaciones en las que me invitaron a compartir parte del trabajo contenido en esta tesis.

Para finalizar, pero no por ello menos importante, quiero recordar a todas las personas que he conocido en el *Laboratorio 216* por haber sido unos compañeros increibles allí dentro y por ser mis amigos fuera. Gracias por "los algoritmos y las algoritmas" pero, sobre todo, por aguantarme, por ayudarme y por escucharme. Gracias, también, a todas las personas que quisieron participar en alguno de mis PFC y a los alumnos la asignatura Lenguajes de Simulación por ayudarme a descubrir lo bonita que puede ser la docencia y lo satisfactorio que es sentir como puedes ser capaz de trasmitir parte de tu conocimiento a otras personas. Ojalá el paso del tiempo me permita recuperar esta actividad de forma continuada.

Llegado un punto de inflexión y decisión en mi vida, como lo es éste, valoro aún más el camino recorrido para llegar hasta aquí. Nada de esto sería posible sin todo aquello que me sostiene y me ha dado fuerza cada día de mi vida. Esta tesis necesitaría un segundo volumen para poder darle las gracias a las personas que me regalan todos y cada uno de los momentos que me hacen sentir alguien infinitamente afortunado por tenerlos ahí y por poder compartir con ellos cada pequeña gran realidad.

Quiero dar las gracias a mi familia por ser quien soy y por haberme enseñado a vivir de esta manera. A mis padres por ser un ejemplo de sacrificio, por haberme facilitado todo lo posible el poder llegar hasta aquí, por haberme permitido hacerlo sin exigencias, por quererme y por aguantarme cada día, con lo difícil que eso es algunas veces. A mi hermano porque las palabras se quedan cortas para decir todo lo que querría, por ser mi principal baluarte y por conseguir darle un sentido diferente y mejor a muchas de las realidades de las que se compone mi vida pasada, presente y futura. A mis abuelos por enseñarme, con su ejemplo, que aquello que uno quiere puede alcanzarse con trabajo y que no hay mayor recompensa que la de sentirse feliz con ello. Ojalá hoy todos pudiéseis vivir este momento, aunque sé que el silencio de unos se confundirá con las palabras de los otros y por un instante todos estaremos aquí. A mis primos por ser mis hermanos, a mis tíos, a mi cuñada que siempre me hace sonreír y a todo el resto de mi gran familia que, aunque repartida por tantos lugares del mundo, hoy no quiero dejar de recordar.

A mis amigos por estar siempre, por escucharme, por ayudarme, por aconsejarme... y por todo. La vida no sería igual sin vosotros, yo no sería igual sin vosotros y nada tendría el sentido que ahora tiene y que me hace sentir realmente ilusionado. Un cachito de todo esto también es vuestro porque yo sólo no habría sido capaz de llegar. Gracias, gracias, gracias a los que estáis, a los que os fuistéis y a los que ya no podréis volver; gracias tanto a los que estáis a este lado del mundo (cerca en Pucela o acompañándome desde muchos otros lugares) como a los que os quedásteis allá en Santiago (y echo un montón de menos).

Gracias a todos porque llegados a este punto me siento emocionado por lo vivido e ilusionado por lo que queda por vivir. Suena Antonio Vega como colofón a estas palabras, "dónde nos llevó la imaginación...", gracias.

Índice general

Índice de figuras

Índice de tablas

Él era sólo un niño
de trenes sin destino,
buscaba su camino
y de repente
algo cambió su vida
en la estación.

Coti Sorokin

1 Introducción

La cantidad de información textual disponible en forma electrónica ha crecido de forma masiva en los últimos años. Un claro ejemplo de esta evolución es la WWW cuyo tamaño ha aumentado desde los 800 millones de páginas, contabilizados en el año 2000 [dMNZBY00], hasta los 20,57 miles de millones en los que actualmente (Mayo, 2010) se estima su tamaño[1]. Además de la WWW, la información textual también prevalece en otros entornos de uso común como las *bibliotecas digitales* o, más genéricamente, las *bases de datos documentales*.

El componente *social* ha sido especialmente relevante en el aumento de la información textual disponible en forma electrónica. La WWW es, de nuevo, un claro ejemplo de ello y, en su interior, proyectos como *Wikipedia*[2] representan como la colaboración social puede dar lugar a la creación de grandes colecciones de texto. Wikipedia surge con el objetivo de crear una enciclopedia que recoja el conocimiento humano y lo haga accesible en la WWW. Por tanto, sus contenidos son un perfecto ejemplo de documentos textuales expresados en lenguaje natural y, por tanto, susceptibles de ser tratados bajo las técnicas de modelado y compresión estudiadas y propuestas en este trabajo de tesis. Además, los contenidos de Wikipedia poseen una segunda propiedad cuya importancia también se analiza en el trabajo actual: el *multilingüismo*.

El asentamiento de los entornos anteriores justifica la existencia de grandes colecciones de texto cuyo tamaño se prevé en continuo aumento. Este hecho plantea la necesidad de disponer de modelos eficientes de representación para estas colecciones que permitan reducir tanto sus costes de almacenamiento como la sobrecarga y retardos asociados a su utilización en procesos de entrada/salida asociada y de comunicación en red. La compresión de texto se postula como una elección idónea ante este escenario.

La *compresión de texto* [BCW90] se centra en encontrar formas de representar un determinado texto en un espacio menor [BYRN99]. Para este propósito, los métodos de compresión identifican regularidades en el texto y, a partir de ellas, obtienen una representación reducida del mismo eliminando la redundancia asociada a dichas regularidades. Esto permite que el texto comprimido ocupe menos espacio de almacenamiento lo que, a su vez, conlleva un importante ahorro temporal en procesos como los anteriormente citados. Además, algunos métodos ofrecen, de forma complementaria, la posibilidad de buscar sobre el texto comprimido, proceso que puede llevarse a cabo hasta 8 veces más rápido que sobre el texto original [dMNZBY00].

El coste al que están sujetos los logros anteriores es, básicamente, el tiempo necesario para la compresión y descompresión del texto. Actualmente, este coste es cada vez menos importante dado el aumento significativo de la velocidad de cómputo de los procesadores respecto a la velocidad de transferencia en disco. Además, el coste de almacenamiento se ha reducido significativamente en los últimos años lo que implica que el ahorro en espacio, por sí mismo, no sea importante en algunos casos. No obstante, el uso de representaciones compactas de los textos permite obtener mejoras significativas en el rendimiento asociado a la utilización de dispositivos

[1] De acuerdo a la estimación referida como *Indexed Web* en http://www.worldwidewebsize.com/
[2] http://www.wikipedia.org

lentos (como discos y canales de comunicación) favoreciendo el uso de la compresión de texto para el manejo de grandes colecciones en los entornos anteriormente referidos.

Los métodos de compresión de carácter universal no plantean restricción alguna sobre los datos que toman en su entrada. Sin embargo, éstos no son capaces de detectar las regularidades específicas de cada tipo de datos, limitando su efectividad en diferentes entornos de aplicación. El presente trabajo de tesis toma como punto de partida el hecho anterior y centra su estudio en la *compresión de textos en lenguaje natural*, entendiendo como tal aquellos textos que contienen información utilizada para la comunicación humana de propósito general. La compresión de textos en lenguaje natural puede ser abordada desde dos perspectivas diferentes. Tradicionalmente se ha considerado el desarrollo de técnicas *ad hoc.* de acuerdo a las propiedades específicas del lenguaje natural. Estas técnicas son capaces de obtener ratios de compresión más competitivas con un conjunto de recursos predeterminado. De forma complementaria, se plantea el diseño de algoritmos de preprocesamiento centrados en obtener representaciones del texto con una redundancia mayor, de tal forma que éstas puedan ser comprimidas de manera efectiva con técnicas universales. Esta segunda alternativa facilita la consecución de soluciones altamente flexibles dada su capacidad para adaptarse a las propiedades específicas de las técnicas universales disponibles para la compresión final del texto.

La naturaleza de la información contenida en este tipo de textos es increíblemente rica y variada, pero también predecible. Un texto en lenguaje natural sigue la estructura impuesta por las reglas sintácticas sobre las que se construye la comunicación humana. Dentro del conjunto de propiedades subyacentes a dicha comunicación se considera la utilización del concepto *palabra* como unidad mínima de información. A partir de esta decisión se desarrollan los conocidos como *métodos de compresión orientados a palabras* capaces de identificar y eliminar la correlación de alto nivel existente entre las palabras de un texto. Considerando la *palabra* como unidad mínima de información, los textos en lenguaje natural están compuestos por símbolos procedentes de dos conjuntos disjuntos: palabras y separadores.

En el presente trabajo se considera que una palabra (ver Definición 3.1, página 34) es una secuencia maximal de caracteres alfanuméricos. Esto supone que un separador es, complementariamente, toda secuencia de caracteres no alfanuméricos (símbolos de puntuación, separación u otros caracteres especiales) existente entre dos palabras consecutivas. Por lo tanto, un texto en lenguaje natural se puede describir como una secuencia de símbolos en la que se *alternan* palabras y separadores [BSTW86]. Un factor importante, dentro de la caracterización anterior, lo representa el hecho de que la mayoría de los separadores, en estos textos, son *espacios en blanco únicos* (según lo indicado en [BYRN99], el 70 %-80 % de los separadores de un texto en inglés son espacios en blanco únicos).

Palabras y separadores presentan distribuciones específicas que sugieren la necesidad de plantear diferentes políticas para su representación. La posibilidad más obvia es la de utilizar un *alfabeto único* en el que se modelen tanto palabras como separadores. Sin embargo, esta política no tiene en cuenta la propiedad de *alternancia* existente entre palabras y separadores. En [BSTW86, Mof89] se plantea la utilización de dos alfabetos específicos: uno para palabras y otro para separadores. La política de *alfabetos separados* está basada en una alternancia en el uso de los alfabetos para la representación de cada símbolo en el texto. De esta manera, conociendo el alfabeto utilizado para la representación del primer símbolo, cada uno de los subsiguientes se representa con el alfabeto complementario al utilizado en el símbolo anterior. Finalmente, Moura, *et al.* [dMNZ97] plantean una variación de esta política. La transformación *spaceless words* considera el espacio en blanco como separador *por defecto* debido a que éste se utiliza de forma mayoritaria después de una palabra. Por lo tanto, si una palabra está seguida por un único espacio en blanco sólo se codifica la palabra, y en otro caso, se codifica tanto la palabra como el separador que la sigue. En el proceso de descompresión, únicamente se decodifica la

palabra y se asume que está seguida por un único espacio en blanco excepto en aquellos casos en los que el siguiente símbolo es un separador. En conclusión, la utilización de esta política descarta el modelado del espacio en blanco, lo que supone la obtención de unas ratios mejores en la compresión de textos en lenguaje natural.

Las propiedades anteriores justifican la utilización de alfabetos de entrada orientados a palabras caracterizados por distribuciones de probabilidad muy asimétricas. Esto supone que un pequeño subconjunto del alfabeto se utiliza de forma muy frecuente mientras el resto de palabras posee una probabilidad de utilización baja. Por lo tanto, los niveles de correlación más altos se identifican entre las palabras y no entre los caracteres individuales, lo cual supone un importante prejuicio para las técnicas de compresión de carácter universal. Sirva como ejemplo el conocido código de `Huffman` [Huf52] (ver Sección §3.3.1). Las tasas de compresión obtenidas por esta técnica evolucionan desde un 65 % (para representaciones del texto orientadas a caracteres) hasta valores próximos al 25 % [MT97] si la codificación se lleva a cabo sobre una representación orientada a palabras. A diferencia de las técnicas universales, los compresores orientados a palabra tienen el problema adicional de gestionar alfabetos de gran tamaño con la eventual complejidad que esto puede aportar tanto en la gestión como en la codificación de sus modelos de representación.

Sin embargo, las técnicas tradicionales de compresión orientada a palabras se desarrollan sobre una base estadística cuya capacidad es limitada a la hora de identificar propiedades semánticas como las subyacentes a aquellos documentos que contienen información almacenada en diferentes idiomas. Dado el trabajo desarrollado en la presente tesis, trataremos el concepto de multilingüismo a través de una perspectiva bilingüe. Nevill-Manning y Bell [NMB92] plantean un primer esquema de compresión para textos paralelos bilingües en los que el mismo contenido semántico se expresa en dos idiomas diferentes. A pesar de lo acertado de su planteamiento, los autores plantean que la mejora de este tipo de propuestas está supeditada a la posibilidad de incorporar un conocimiento complementario derivado del uso de técnicas relacionadas con la traducción automática. Conley y Klein [CK08] incorporan este conocimiento a través de técnicas de *alineamiento* que transforman el texto original en una representación equivalente en el que la relación de traducción se plantea de forma explícita.

Los textos paralelos bilingües (*bitextos*) aportan una fuente importante de conocimiento lingüístico dado que la información expresada en cada uno de los idiomas puede ser observada como una anotación del significado de la información expresada en el idioma complementario [Mel01]. Esta propiedad favorece la utilización de los bitextos en diferentes contextos de aplicación tanto de naturaleza monolingüe (por ejemplo en aplicaciones que necesiten desambiguación de palabras) como aquellas que explotan el carácter bilingüe de la información contenida en el bitexto (como la traducción automática o la recuperación de información translingüe).

La naturaleza *on-line* de algunas de estas aplicaciones plantea la necesidad de disponer de mecanismos que permitan utilizar de forma eficiente, en espacio y tiempo, la información contenida en el bitexto. En lo que respecta a la eficiencia espacial, un bitexto almacena por duplicado la misma información, dado que ésta se expresa en dos idiomas diferentes. De nuevo la compresión de texto aparece como la solución natural para este problema. Sin embargo, para el caso actual se precisa de un esquema que permita representar tanto la información propiamente contenida en el bitexto como aquella que se necesita para obtener su representación en cada uno de los idiomas. En lo relativo a la eficiencia temporal esperada en la compresión de bitextos, es importante reseñar que la mayoría de las aplicaciones que hacen uso de ellos no requieren una descompresión completa del contenido sino que su necesidad básica se centra en la posibilidad de utilizar de forma directa la información almacenada en el bitexto. Además, es deseable que el acceso a la información pueda llevarse a cabo de acuerdo a las propiedades de la aplicación que la utiliza. Esto supone soportar accesos monolingües (la consulta de información y

el resultado obtenido se expresan en el mismo idioma) y translingües (la consulta de información y el resultado se expresan en idiomas diferentes) al bitexto comprimido.

El presente trabajo de tesis plantea sus objetivos en el contexto anterior. La siguiente sección presenta una revisión de dichos objetivos que sugieren una conceptualización del trabajo llevado a cabo y cuyos resultados suponen un conjunto de aportaciones presentadas en la Sección §1.2. Finalmente, la Sección §1.3 describe la estructura de este documento.

1.1 Objetivos

La compresión de texto en lenguaje natural sugiere, cómo se ha explicado previamente, la utilización de modelos de representación orientados a palabra con los que poder identificar la redundancia inherente a la estructura de palabras y separadores que los define de acuerdo a la perspectiva humana con la que se obtienen. El presente trabajo de tesis enfoca el diseño de técnicas de compresión orientadas a palabra capaces de aprovechar la correlación de alto nivel existente entre las unidades componentes de un texto. Para este propósito se consideran sendas perspectivas basadas en el modelado del texto sobre estadísticas de orden superior capaces de obtener una mejor representación de la correlación existente entre las palabras:

- El objetivo inicial se centra en incorporar un conjunto de heurísticas identificadas en lenguaje natural a un esquema de preprocesamiento del texto basado en un diccionario de palabras. Este esquema no desarrolla, propiamente, un rol de compresión sino que sólo transforma el texto en una representación equivalente más redundante que la original. Dicha representación se utiliza, posteriormente, como entrada al proceso de compresión llevado a cabo con técnicas universales. La capacidad compresiva de este tipo de propuestas radica en que el texto transformado ofrece una distribución de los símbolos más sesgada que la original de tal forma que una técnica universal es capaz de obtener una identificación más efectiva de las regularidades subyacentes al texto y, por consiguiente, mejorar sus tasas de compresión e incluso acelerar, en algunos casos, sus procesos de compresión y descompresión.

- El desarrollo de una técnica *ad hoc.* para compresión de lenguaje natural complementa el objetivo anterior. Para ello se define un modelo del lenguaje que representa la relación de adyacencia, existente entre las palabras, a través de una estructura de grafo dirigido. Los nodos almacenan las palabras mientras que las aristas representan la relación de adyacencia sobre la que se fundamenta la técnica actual. Esta representación plantea un modelo de orden 1 orientados a palabras. La experiencia obtenida con esto modelo se utiliza como base para la obtención de un modelo de orden superior. Este caso se centra en evaluar cómo una gramática libre de contexto puede servir como estructura de representación para la jerarquía existente entre los contextos utilizados en un modelo de orden superior. Esta representación configura un modelo de orden 1 orientado a frases que puede ser utilizado en un entorno práctico de ejecución en tiempo de cómputo y espacio en memoria. El desarrollo de la gramática se fundamenta en la experiencia obtenida en el algoritmo `Re-Pair` [Lar99, LM00] cuyo funcionamiento se adapta a las necesidades planteadas en el problema actual.

El conocimiento y la experiencia adquirida en este contexto realimenta la consecución de los objetivos asociados a la en la representación compacta de textos bilingües. Como se planteaba en la Introducción, este tipo de textos posee propiedades estadísticas comparables a las de un texto genérico en lenguaje natural, por lo que la experiencia anterior se utiliza como punto de partida para su compresión.

- Inicialmente se analizan las posibilidades que ofrecen las técnicas de compresión orientadas a palabra para la incorporación de mecanismos específicos para la gestión y representación de la relación de traducción mutua subyacente al bitexto. Nótese como en el caso actual no sólo interesa alcanzar una mayor efectividad en la compresión sino que además se busca la posibilidad de obtener acceso monolingüe y translingüe al texto comprimido. De acuerdo con estos objetivos se considera la definición del concepto de *bipalabra*, como una unidad simbólica compuesta por dos palabras que representan una traducción del mismo concepto en los dos idiomas del bitexto. Por lo tanto, esta segunda perspectiva se aproxima a una compresión conceptual del bitexto en la que las palabras procedentes de ambos idiomas se representan en el ámbito que plantea cada bipalabra. La jerarquización obtenida a partir de la relación palabra-bipalabra ofrece una solución apropiada para el diseño de operaciones de búsqueda, basadas en *pattern-matching*, sobre el texto comprimido. Por lo tanto, el objetivo final, en lo que respecta a la representación compacta de bitextos, se centra en explotar las posibilidades de búsqueda y en su incorporación a aplicaciones reales como, por ejemplo, la búsqueda de concordancias bilingües.

1.2 Aportaciones

Como se plantea en la sección anterior, el presente trabajo de tesis afronta dos objetivos principales:

- La definición de nuevas técnicas de compresión orientadas a palabra basadas en la utilización de estadísticas de orden superior deducidas de las propiedades inherentes a la estructura del lenguaje natural.

- El estudio y definición de técnicas de compresión para textos paralelos bilingües que permitan su utilización en diferentes contextos de aplicación de carácter mononlingüe y bilingüe.

cuyas aportaciones se revisan de forma independiente.

Compresión de Lenguaje Natural. Las técnicas propuestas para texto en lenguaje natural cubren las dos perspectivas indicadas por Skibiński, *et al.* [SGD05].

Word-Codeword Improved Mapping (`WCIM`) plantea un algoritmo de preprocesamiento basado en diccionario que transforma el texto original en una representación equivalente orientada a byte. Esta transformación reduce a, aproximadamente, 2 bytes (al utilizar codificación `ETDC` [Far05, BFNP07]) la longitud media de los códigos utilizados para la representación de cada palabra, cuya longitud media original es de, aproximadamente, 5 caracteres para textos en inglés [BCW90]. Esta propiedad posibilita la identificación de correlaciones de un orden superior a las que la técnica original es capaz de detectar para un orden de modelo determinado. Por consiguiente, la transformación `WCIM` permite obtener representaciones altamente compresibles con técnicas basadas en propiedades diferentes: $\texttt{WCIM}_{ppmdi}$, $\texttt{WCIM}_{p7zip}$ y $\texttt{WCIM}_{bzip2}$ se muestran como soluciones efectivas basadas, respectivamente, en una compresión predictiva con `PPM` [CW84b], en un modelo de diccionario como `lzma`[3] y en el uso de la transformada de *Burrows-Wheeler* [BW94]. El caso de `lzma` es especialmente significativo, también para la técnica original `mPPM` [AdlF06] (en el que se basa la propuesta `WCIM` actual) cuya efectividad es significativamente mejor que la presentada por sus autores sobre compresión `PPM`.

La transformación `WCIM` comprende cuatro heurísticas específicas para la gestión de diccionarios de gran tamaño como el requerido en lenguaje natural. Nótese que la frecuencia de los

[3] http://www.7-zip.org/

símbolos utilizados en este tipo de textos sigue una distribución *power-law* que puede ser caracterizada de acuerdo a las leyes de Heaps [Hea78, BYRN99] y Zipf [Zip49, BYRN99] (ver Sección §3.1). Esto supone que la experiencia actual sobre `WCIM` puede ser utilizada en otros contextos de aplicación caracterizados, igualmente, sobre distribuciones *power-law*. Actualmente, la heurística definida para la gestión de símbolos "unitarios" (símbolos con una única ocurrencia) ha sido incorporada en las técnicas `TRC` planteadas para la compresión de textos bilingües (ver Sección §6.4.3) y probada con éxito en la compresión de grafos `RDF`[4].

Por su parte, las técnicas *Edge-Guided* (`E-G`) definen una familia de compresores orientados a palabra basados en una representación del texto sobre un modelo de orden superior. La técnica base $\texttt{E-G}_1$, construida sobre un grafo dirigido de palabras, valida el modelo de representación propuesto y obtiene unas tasas de compresión competitivas para el esquema de codificación diseñado. Además, afronta con éxito el problema principal de los compresores de orden superior orientados a palabra: los costes (en espacio y tiempo) derivados de la gestión de un modelo complejo y el tamaño (en bits) necesario para su representación en el texto comprimido. Esta experiencia inicial asienta las bases para la extensión del orden del modelo en la técnica $\texttt{E-G}_k$. La principal aportación de este compresor radica en la utilización de una gramática libre de contexto para la representación de la jerarquía de niveles requerida en el modelo. La gramática, tipo `Re-Pair` [Lar99, LM00], se añade al modelo original a través de una relación de jerarquía que permite derivar las *heavy paths* [HT84] formadas por los componentes derechos de cada regla. Esto permite evaluar, cada uno de estos componentes, como posibles contextos de representación de un determinado símbolo una vez se ha comprobado que éste no puede ser codificado en el nivel justo superior. El modelo $\texttt{E-G}_k$ resultante satisface los requisitos iniciales, obteniendo una implementación práctica (en tiempo y memoria) con una efectividad competitiva de hasta el 18 % para textos de mediano-gran tamaño.

Compresión de Textos Bilingües. Las propuestas para la compresión de bitextos planteadas en la presente tesis complementan las previamente existentes [NMB92, CK08] en un contexto de investigación poco explorado hasta el día de hoy. En primer lugar, las técnicas propuestas se integran correctamente con diferentes políticas de alineamiento textual aprovechando de forma efectiva las propiedades subyacentes a cada una de ellas. Esto permite obtener compresores orientados a palabra (*Translation Relationship-based Compressor*: `TRC`) y bipalabra (*Two-Level Compressor for Aligned Bitexts*: `2LCAB`) capaces de obtener unas tasas de compresión competitivas. Complementariamente, resulta reseñable la integración de la técnica $\texttt{E-G}_1$ (obtenida genéricamente para la compresión de lenguaje natural) en el compresor $\texttt{TRC}_{(1;1)}$ para textos bilingües. Esta solución es la que muestra una mejor efectividad de todas las propuestas planteadas en el Capítulo 6.

Además, el modelado *conceptual* obtenido sobre la definición de bipalabra facilita la consecución, con `2LCAB`, de la primera representación comprimida de bitextos que permite consultar y manipular directamente su contenido a través de las siguientes operaciones:

- `2LCAB` permite identificar todas las traducciones asociadas con una palabra determinada sin necesidad de acceder al bitexto comprimido.

- Facilita la localización de todas las ocurrencias de una palabra dada mediante procesos de *multi pattern-matching* exacto sobre el bitexto comprimido. De acuerdo a la relación de traducción representada en el diccionario de bipalabras, la búsqueda anterior también permite localizar las ocurrencias de todas las traducciones, en el idioma complementario, de la palabra solicitada.

[4] http://www.w3.org/RDF/

- `2LCAB` es capaz de desambiguar la búsqueda de una palabra a través del propio concepto de bipalabra. Esto es, dada una palabra en un idioma, su significado puede ser anotado por una palabra del idioma complementario localizando sólo las ocurrencias de la palabra en las que ésta tenga el significado deseado.

- Finalmente, `2LCAB` permite la descompresión parcial del bitexto, permitiendo con ello la extracción eficiente de *snippets*. Dicha extracción se puede llevar a cabo tanto en el texto del idioma en el que se ha expresado el patrón de búsqueda como en el complementario.

El conjunto de operaciones anteriores permite la utilización de `2LCAB` como motor de búsqueda en aplicaciones genéricas basadas en concordancias bilingües. Asimismo, la experiencia obtenida con `2LCAB`, y sus posibilidades de búsqueda orientada a palabras, plantea el primer paso hacia la obtención de una representación comprimida de los bitextos alineados con otras políticas de mayor complejidad. Este tipo de representaciones poseen unas características comparables a los modelos utilizados en traducción automática con la notable diferencia de que su representación comprimida aumentará significativamente su eficiencia respecto a este tipo de modelos. Esto supondrá un paso adelante para la incorporación de los bitextos a aplicaciones de naturaleza *on-line* como, por ejemplo, en sistemas de recuperación de información multilingües que precisan de una traducción al vuelo de las consultas suministradas por los usuarios.

1.3 Estructura de la tesis

El presente documento está estructurado en *siete* capítulos, incluyendo el actual. Como se ha planteado hasta aquí, el presente capítulo introduce el contexto en el que se desarrolla este trabajo de tesis incidiendo en sus objetivos básicos y en las aportaciones obtenidas a partir de su realización.

Los capítulos 2 y 3 plantean una revisión del "estado del arte" relacionado con trabajo actual. En primer lugar, el capítulo 2 revisa los *Conceptos Básicos*, de Teoría de la Información, relacionados con la compresión de datos. En este repaso se definen los conceptos de *información* y *entropía de una fuente de información* como paso previo al estudio de las propiedades de la desigualdad de Kraft, básicas para la codificación de un mensaje. Esta organización del capítulo permite concretar los aspectos generales de las etapas de modelado y codificación que conforman, en su conjunto, la descripción genérica de una técnica de compresión. El capítulo finaliza con una breve clasificación de las técnicas de compresión de datos. Esta clasificación representa la conceptualización básica sobre la que se extienden los conceptos contenidos en el capítulo 3 relativo a la *Compresión de Texto*. Este capítulo comienza estudiando algunas de las leyes fundamentales que explican las propiedades básicas del lenguaje natural sobre las que diseñan los algoritmos de compresión para texto. A partir de esta caracterización se detallan las etapas de modelado y codificación de una fuente de información textual. En lo que respecta a la primera etapa, se detallan las propiedades de los modelos adaptativos y semi-estáticos incidiendo, en este último caso, en las propiedades que permiten un acceso y una búsqueda directa (con algoritmos de *pattern-matching*) sobre el texto comprimido con estas técnicas. La etapa de codificación aborda la descripción de las técnicas clásicas, prestando especial atención a los códigos orientados a byte que serán utilizados como base de algunas de nuestras propuestas. El capítulo finaliza repasando las técnicas de compresión más conocidas siguiendo la clasificación ofrecida en el capítulo 2.

Los tres capítulos siguientes presentan cada una de las propuestas planteadas en el presente trabajo de tesis. Todos ellos muestran una estructura similar en lo que respecta al desarrollo de sus contenidos. En primer lugar se presenta el trabajo relacionado con cada una de las propuestas así como aquellos conceptos que puedan ser considerados, en cada caso, como básicos para el

seguimiento del capítulo. A continuación se estructuran los contenidos asociados a cada propuesta, que suponen su conceptualización y descripción previa a su experimentación con colecciones textuales de referencia. Cada uno de los capítulos contiene una sección de experimentos en los que se analizan, de forma empírica, las propiedades de cada una de nuestras propuestas. Los capítulos finalizan con una última sección de conclusiones y trabajo futuro donde se muestra un análisis de lo planteado en dicho capítulo y las posibilidades de trabajo futuro consideradas como extensión de lo actualmente presentado.

El capítulo 4 presenta la primera de las propuestas planteadas en esta tesis. `WCIM` define un algoritmo de preprocesamiento (basado en un diccionario de palabras) que contempla la utilización de diferentes heurísticas obtenidas sobre propiedades estadísticas identificadas en el texto. La efectividad de `WCIM` como técnica de compresión se basa en que el algoritmo de preprocesamiento obtiene una representación más compresible del texto utilizando, para ello, una codificación orientada a byte que posteriormente se comprime con una técnica universal orientada a bit. `WCIM` toma, como punto de partida, la experiencia obtenida en una propuesta previa (referida como `mPPM` [AdlF06]) y se apoya fuertemente en las conclusiones presentadas por Fariña, *et al.* [FNP08] acerca de la compresibilidad de las representaciones del texto preprocesado. `WCIM` transforma el texto original en una representación equivalente orientada a byte sobre `ETDC` (la Sección §3.3.3 plantea un resumen de las técnicas basadas en códigos densos, refiriendo adecuadamente cada una de ellas) y considera técnicas universales, con diferentes propiedades, para la compresión final del texto preprocesado.

El capítulo 5 plantea la técnica `Edge-Guided` (`E-G`), centrada en la construcción y codificación de un modelo del texto basado en la información contenida en las relaciones de adyacencia existentes entre sus unidades constituyentes. Dependiendo del nivel de granularidad con el que se eligen estas unidades (alfabeto de entrada) se obtienen sendas técnicas referidas como `E-G`$_1$ y `E-G`$_k$. La técnica `E-G`$_1$ (considerada como caso base) plantea una representación del texto orientada a palabras sobre un grafo dirigido. Cada uno de los nodos i del grafo almacena una palabra p_i identificada en el texto y, a su vez, se utiliza como contexto para la representación y codificación de aquellos nodos j que almacenan palabras p_j adyacentes a p_i en el texto. De esta manera, las aristas (i, j) se añaden al grafo siempre que la secuencia de palabras $p_i p_j$ aparezca contigua en el texto. Cada uno de los nodos se maneja y codifica, de forma independiente, mediante una técnica basada en diccionario. Esto requiere la utilización de heurísticas que faciliten tanto un manejo eficiente de la información almacenada en los nodos como una codificación efectiva de la misma. La técnica `E-G`$_k$ generaliza el caso anterior extendiendo el alfabeto de entrada con secuencias de palabras significativas de acuerdo a sus propiedades en el grafo de representación. Para este propósito se hace uso de una gramática libre de contexto que posteriormente se añade al modelo de grafo inicial. Esta decisión permite combinar, en un mismo modelo, contextos de diferente orden obteniendo estadísticas de orden superior en la representación conseguida. Las técnicas `E-G` poseen propiedades interesantes para el preprocesamiento textual, sin embargo alcanzan sus cotas de efectividad más altas cuando se utilizan directamente con una codificación de `Huffman` (ver sección 3.3.1) orientada a bit.

El capítulo 6 enfoca la compresión de textos bilingües a través del concepto de corpus paralelo (*bitexto*). De forma genérica, un bitexto contiene información expresada en lenguaje natural con la salvedad de que esta información representa el mismo contenido semántico en dos idiomas diferentes. Esta caracterización plantea una doble interpretación: 1) un bitexto posee las mismas propiedades que un texto en lenguaje natural con la diferencia de que éstas se distribuyen de acuerdo a la naturaleza particular de cada uno de los idiomas utilizados; 2) un bitexto contiene una gran cantidad de redundancia semántica bajo la consideración de que expresa, por duplicado, la misma información. Las diferentes técnicas de representación y codificación propuestas en este capítulo parten de bitextos previamente alineados que contienen la misma

información que el texto original pero representada de tal forma que la relación de traducción puede ser utilizada con el fin de identificar y eliminar la redundancia semántica existente. La compresión de bitextos conlleva, en primer lugar, la definición de modelos específicos para su representación. Para este propósito se adoptan sendas perspectivas. Por un lado se plantean modelos orientados a palabras basados en la naturaleza del bitexto como lenguaje natural y en la relación de traducción aportada en su representación alineada. Por otro lado se define un nuevo símbolo, denominado *bipalabra* [MPASM$^+$09], que integra en una entidad única sendas palabras identificadas como traducción mutua en el bitexto. A partir de estos modelos se construyen sendos compresores orientados a palabra (`TRC`) y bipalabra (`2LCAB`) con diferentes propiedades ante el texto comprimido. Nótese que la técnica `2LCAB` (basada en codificación `ETDC`) posee propiedades de acceso y búsqueda directa al texto comprimido lo que permite realizar operaciones específicas que permiten localizar contenidos en el bitexto con independencia del idioma en el que se realice la consulta y en el que se solicite el resultado. Las diferentes propuestas se muestran competitivas tanto en contextos que demandan unas mayores tasas de compresión como aquellos que precisan un acceso y manipulación eficiente de la información contenida en el bitexto.

Finalmente, el capítulo 7 complementa las conclusiones planteadas en los tres capítulos anteriores, ofreciendo una revisión global de los objetivos conseguidos. Además ofrece un resumen de algunas de las ideas de trabajo futuro consideradas a raíz de los resultados previamente obtenidos y de las posibilidades identificadas para su evolución y mejora. Complementando los contenidos anteriores, la sección finaliza mostrando una revisión de las publicaciones llevadas a cabo sobre los resultados presentados en los capítulos 4, 5 y 6 del presente documento.

Pero como explicar
que me vuelvo vulgar
al bajarme de cada escenario.
Enrique Urquijo

2
Conceptos Básicos

La compresión de datos se plantea, originalmente, como una disciplina centrada en la reducción del número de bits utilizados para la representación de un determinado contenido. Esto no sólo permite conseguir un ahorro en el espacio de almacenamiento sino que también hace más eficientes los procesos de acceso a disco y de transmisión en red.

El aumento progresivo de la capacidad de almacenamiento de los nuevos dispositivos ha ido reduciendo el interés en el ahorro espacial, por sí mismo, para centrarse en el impacto que éste supone en la mejora de eficiencia que puede ser obtenida al operar con el mensaje comprimido. En los últimos años se ha mantenido la tendencia indicada por Ziviani, *et al.* [ZdMNBY00] una década atrás; esto supone que la velocidad de cómputo de los procesadores ha seguido creciendo más rápido que la velocidad de transferencia en disco de forma que el tiempo de descompresión se puede amortizar con la ganancia obtenida en el tiempo de transferencia del contenido comprimido respecto al contenido original. Los objetivos de la compresión de datos se expanden, sobre estas expectativas, hacia técnicas más eficientes que aprovechen estas propiedades y consigan formas de representación que lleguen, incluso, a permitir la operación directa sobre el mensaje comprimido

Los sistemas de compresión se pueden catalogar en dos familias independientes: compresores sin pérdida de información (*lossless*) y compresores con pérdidas de información (*lossly*). Estos últimos tienden a obtener unas mejores tasas de compresión dado que la información que se descomprime de ellos no coincide, exactamente, con la originalmente comprimida. La aplicación de los compresores con pérdida se lleva a cabo sobre datos cuya naturaleza permite renunciar a parte de su información sin que esto suponga una pérdida cualitativa en el mensaje transmitido. Básicamente, la compresión con pérdida se aplica a datos de naturaleza audio-visual (imágenes, vídeo o sonido) dado que parte de su información no afecta, sustancialmente, a la recepción de su mensaje por parte de un humano. Sin embargo, esta pérdida no es aceptable en otras áreas de aplicación en las que se precisa que el contenido descomprimido coincida exactamente con el mensaje original, ya que una variación podría impedir el entendimiento del contenido transmitido. Consecuentemente, este es el tipo de compresión deseada al tratar con ficheros ASCII, registros de una base de datos o secuencias biológicas, entre muchas otras aplicaciones.

El capítulo actual afronta una revisión de los conceptos básicos de teoría de la información [Sha48, Abr63] sobre los que se desarrolla el área de compresión de datos. En primer lugar la definición de *información* (§2.1) presenta, desde la base, el contexto en el que se encuadra este trabajo de tesis. La Sección §2.2 define los conceptos básicos de *redundancia* y *compresión* antes de afrontar, directamente, los conceptos relacionados con una fuente de información, su modelado y codificación (§2.3). La Sección §2.4 plantea el concepto de *entropía* y lo estudia en modelos de diferente orden sobre los que se analiza su capacidad representativa. Por su parte, la Sección §2.5 estudia las propiedades básicas de los códigos desde una perspectiva ajustada al contexto actual. Las secciones finales plantean, respectivamente, una clasificación de los diferentes métodos de compresión (§2.6) incidiendo en algunos aspectos significativos desde la perspectiva de la compresión de texto tratada en el capítulo siguiente y una caracterización genérica tanto de la efectividad (§2.7) como de las limitaciones (§2.8) de los métodos de compresión.

2.1 Definición de información

Aunque el concepto de *información* es bastante general, su definición resulta compleja dada la ambigüedad existente en torno a su propia interpretación. Desde una perspectiva genérica, información es todo aquello que proporciona algún significado a través del mensaje que transmite. Esta es la misma definición que se acepta en el contexto informático en el que se desarrolla el presente trabajo. Únicamente se puede añadir que esta información requiere algún tipo de soporte que facilite su representación tanto para su transmisión como para su almacenamiento. Integrando todo lo anterior, se acepta que la información se almacena o transmite mediante mensajes. Un mensaje contiene información siempre que éste aporte algo nuevo o imprevisto; por lo tanto, la cantidad de información contenida en un mensaje está directamente relacionada con su aportación cognitiva.

Un proceso genérico de transmisión de información precisa tres elementos básicos: el *emisor* (o fuente) *de información*, el *canal de comunicación* y el *receptor* (o destino) *de información*. El proceso de transmisión es sencillo, partiendo del emisor y alcanzando al receptor a través del canal de comunicación, dentro del cual la información puede ser alterada en el caso de que éste contenga ruido [Abr63, CT91]. En el contexto de la compresión de datos se asume, en la mayoría de los casos, que el canal de comunicación no posee ni ruido ni memoria. Sobre estas premisas, la definición de *información asociada* a un evento o suceso transmitido por un canal sin ruido es la siguiente:

Definición 2.1 (Información asociada) *La información asociada al evento x_i, con una probabilidad de ocurrencia p_i, en un canal sin ruido ni memoria, se puede calcular mediante la expresión:*

$$I(x_i) = \log_2 \frac{1}{p_i} = -\log_2 p_i \tag{2.1}$$

De esta definición se puede deducir que:

- Si el evento a ocurre siempre, el proceso de comunicación no proporciona información alguna al receptor.

$$p(a) \to 1 \iff I(a) \to 0$$

- Si la probabilidad de ocurrencia de a es un valor pequeño, próximo a 0, el proceso de comunicación proporciona una alta información al receptor.

$$p(a) \to 0 \iff I(a) \to 1$$

2.2 Redundancia y compresión

Los sistemas de compresión operan sobre las premisas básicas de identificar y eliminar la redundancia existente en la información contenida en el mensaje. Una compresión efectiva es capaz de reducir el tamaño del mensaje original frente al tamaño resultante del mensaje comprimido. La redundancia encontrada en la gran mayoría de las secuencias de datos es de naturaleza *estadística*. Esto supone que aquellas secuencias que no presentan esta redundancia (referidas como secuencias aleatorias) están formadas a partir de un conjunto de símbolos distribuidos uniformemente sobre la misma probabilidad de ocurrencia. En definitiva, la redundancia estadística se traduce en una distribución de probabilidad no uniforme en la que cada símbolo utilizado por la fuente de información se caracteriza por una probabilidad de ocurrencia particular.

El tipo y la cantidad de redundancia existente en un mensaje depende de la naturaleza de los datos que contiene y caracteriza fuertemente las propiedades del compresor y el descompresor. Existen tres tipos de redundancia:

Redundancia en la codificación: los compresores centrados en la eliminación de este tipo de redundancia se basan en que el alfabeto utilizado en la fuente de información está caracterizado por un una distribución de probabilidad no uniforme. Esto supone que cada símbolo tiene una probabilidad de aparición particular de forma que algunos símbolos ocurren con mayor frecuencia que el resto. Sobre esta premisa, `Huffman` [Huf52] propone un algoritmo de compresión basado en un código de longitud variable (*variable-length coding*) que afronta esta redundancia favoreciendo una representación más efectiva de los símbolos más frecuentes. El resultado de este algoritmo es una representación comprimida a partir del cual se puede obtener, sin pérdida, el mensaje original.

Redundancia en la secuencia de aparición: los compresores que identifican y eliminan este tipo de redundancia consideran la existencia de correlación entre los símbolos previamente utilizados y los siguientes. Esta redundancia puede observarse, de forma sencilla, en un texto. Por ejemplo, en castellano la probabilidad de que la letra `m` preceda a `p` es mucho mayor que la probabilidad de que sea la letra `n` la que preceda a `p`. Este tipo de correlaciones, derivadas de las reglas sobre las que se construye el mensaje, añaden una redundancia que puede ser detectada y eliminada como hacen compresores sin pérdida, tan diferentes en su conceptualización, cómo `RLE`, `LZW` o `PPM`.

Redundancia psico-visual o psico-auditiva: este tipo de redundancia se caracteriza sobre la consideración de que la capacidad sensorial de un humano es limitada en lo que respecta a la percepción de alguna información. Por lo tanto, la detección de esta redundancia se lleva a cabo sobre fuentes de información de naturaleza audiovisual. La redundancia psico-visual o psico-auditiva puede ser eliminada del mensaje sin que la pérdida de información suponga un deterioro sensible en la interpretación humana de la imagen, audio o vídeo contenido en el mensaje. Esta decisión permite conseguir altas tasas de compresión a costa de la pérdida irreversible de parte de la información contenida en el mensaje original. Los algoritmos de compresión basados en transformadas [JN84, Kou95] como la del coseno, la transformada rápida de Walsh-Hadamard o la transformada de Karhunen-Loève se centran en la eliminación del presente tipo de redundancia.

2.3 Comprimir = Modelar + Codificar

Cualquier proceso de compresión puede observarse a partir de la integración de dos etapas complementarias: *modelado* y *codificación*. Esta perspectiva se deriva de la forma en la que Shannon afrontó el cálculo de la entropía media del idioma inglés [Sha51]. Para este propósito, consideró como predictores un conjunto de personas cuyo idioma nativo era el inglés y midió la entropía existente en el error de la predicción estimando que, aproximadamente, el 50 % de los caracteres utilizados en inglés son redundantes[1]. Sobre esta experiencia, Rissanen y Langdom [RL81] concretan que una técnica de compresión se descompone en dos partes complementarias: (1) un modelo de la fuente a comprimir y (2) un codificador para dicho modelo (ver figura 2.1).

[1] La conclusión de este estudio sugiere que la misma información podría ser expresada con la mitad del mensaje actual dado que el resto puede ser predicho a partir de la propia estructura del lenguaje.

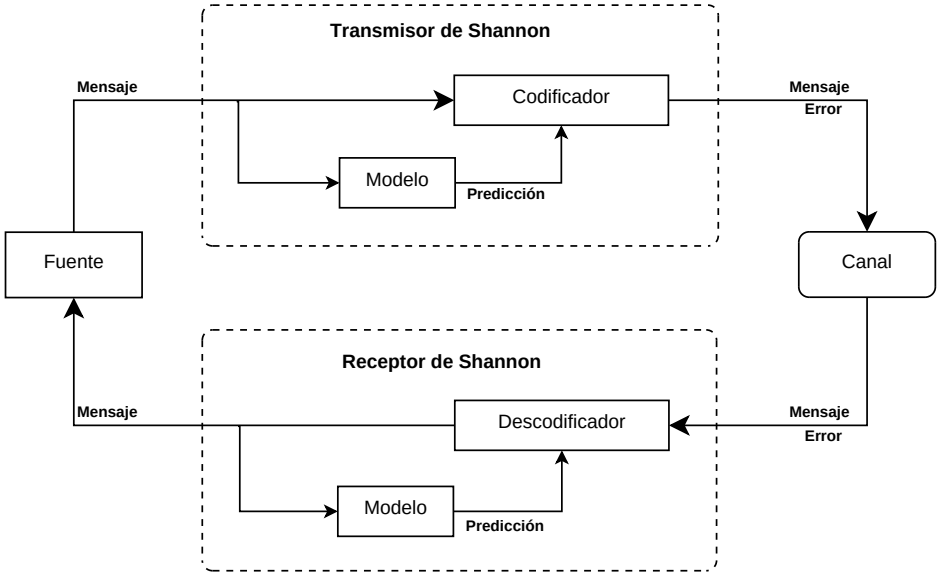


Figura 2.1: Paradigma de la compresión/descompresión formulado por Shannon.

Modelado. Inicialmente, el modelado de una fuente de información requiere, definir qué se considera como *símbolo*, de acuerdo a sus propiedades específicas. Un símbolo es, generalmente, un carácter único, un número finito de caracteres o una palabra de texto definida como una secuencia maximal de caracteres alfanuméricos o no alfanuméricos. El conjunto de todos estos símbolos se denomina *alfabeto fuente*. Los mensajes emitidos por una fuente de información están formados por la combinación de los símbolos contenidos en dicho alfabeto fuente. Aunque cada mensaje tiene unas características particulares, las reglas generales consideradas por la fuente de información facilitan la identificación de regularidades genéricas en los mensajes. Este es el objetivo principal del modelo: obtener una representación estadística del mensaje que facilite el aprovechamiento de estas regularidades en la mejora de la efectividad del compresor.

Codificación. La etapa de codificación establece la representación, en el mensaje comprimido, de cada símbolo modelado de acuerdo a sus propiedades en el mensaje original [BWC89]. El objetivo principal del codificador es encontrar una representación efectiva de la información que el modelo no es capaz de predecir o representar implícitamente. Básicamente, cada símbolo emitido en la fuente genera un nuevo símbolo en el modelo, de forma que la discrepancia existente entre ambos es lo que el codificador representa de una forma compacta.

Esta sencilla caracterización, en dos etapas, muestra como un compresor requiere obtener un modelo de la fuente sobre el que posteriormente aplicar una codificación efectiva para la información que éste representa. La combinación de ambas etapas da lugar a numerosas técnicas de compresión, específicas para diferentes fuentes de información y basadas en técnicas de codificación particulares que aportan unas propiedades determinadas al resultado obtenido con cada una de ellas. A continuación se profundiza en las propiedades de una fuente de información.

2.3.1. Fuentes de información

Una fuente de información discreta emite mensajes basados en la sucesión de símbolos procedentes de un conjunto finito de eventos denominado *alfabeto fuente* ($\mathcal{A}$). La notación

$$\mathcal{A} = \{x_1, \ldots, x_\sigma\}$$

refiere un alfabeto fuente $\mathcal{A}$ compuesto por σ símbolos x_i. El valor σ se denomina *base* de la fuente de información discreta.

En el ámbito de la teoría de la información, se considera que la emisión de símbolos por parte de una fuente de información responde a un proceso estocástico. Esto supone que la emisión de símbolos en una fuente de información se caracteriza de acuerdo a las probabilidades de ocurrencia de los eventos, $p(x_i) = p_i$, que son conocidas:

$$\mathcal{P} = \{p_1, \ldots, p_\sigma\} \ / \ \sum_{i=1}^{\sigma} p_i = 1 \ \wedge \ \forall p_i \geq 0$$

La cantidad de información recibida en m mensajes, procedentes de una fuente con n posibles, debe ser la misma que al recibir *un* único mensaje procedente de una fuente que puede emitir un total de n^m mensajes. Esta propiedad sugiere que la cantidad de información contenida en un mensaje sigue una función logarítmica. Dadas dos fuentes equiprobables de n y n^m mensajes, respectivamente, la información obtenida al recibir un mensaje de la primera fuente es $m \times f(n)$, donde f es una función creciente (nótese que la probabilidad de cualquier mensaje es $1/n$ al tratarse de una fuente equiprobable; sin embargo, f es función de la inversa de la probabilidad lo que la hace dependiente de n). Por su parte, la recepción de un mensaje de la segunda fuente aporta una información $f(n^m)$. Esto supone que la naturaleza logarítmica de la información puede ser apreciada bajo la consideración de que ambos mensajes transmiten la misma información $m \times f(n^m)$.

El concepto de *entropía*, propuesto por Shannon [Sha48], contrasta estas propiedades sobre un teorema que establece la relación entre las probabilidades de los símbolos y los códigos utilizados para su representación. Dicho teorema demuestra que la representación más efectiva de un símbolo con probabilidad p requiere $-\log_2 p$ bits. Esto supone que los símbolos que presentan una alta probabilidad de aparición se codifican con un menor número de bits mientras que los símbolos menos frecuentes requieren un mayor número de bits para su representación.

2.4 Modelado de una fuente de información

Si consideramos un muestreo lo suficientemente grande de un mensaje emitido por una determinada fuente de información, se cumple que la muestra obtenida es representativa (desde una perspectiva estadística) del conjunto de todas las cadenas de Markov[2] que puede emitir dicha fuente. Esta caracterización garantiza que se trata de un proceso *ergódico* ya que la muestra es estadísticamente representativa de toda la información que puede emitir la fuente analizada.

Los *modelos de contexto finito* caracterizan la probabilidad de cada símbolo de acuerdo a la secuencia finita de símbolos que lo preceden. El conjunto finito de símbolos anteriores, a uno dado, ofrece un buen contexto de predicción apoyándose en el hecho de que si un determinado símbolo ha aparecido tras un determinado contexto existe una alta probabilidad de que lo vuelva a hacer.

Definición 2.2 (Contexto de un símbolo) *El contexto de un símbolo es la secuencia de símbolos, de tamaño finito, que lo precede.*

[2]Una *cadena de Markov* se define como una secuencia de símbolos (eventos) en la que la probabilidad de aparición de cada uno de estos símbolos depende exclusivamente de la secuencia finita que que la precede.

Los modelos de contexto finito se basan en un análisis de frecuencias de los q-gramas (grupos de q símbolos consecutivos) identificados en el mensaje original. La frecuencia de estos q-gramas se utiliza para construir modelos en los que los primeros $q-1$ símbolos se usan como predictores del q-ésimo. Generalmente, el contexto formado por los símbolos más próximos es el que permite obtener una mejor predicción, considerando que al aumentar el tamaño de dicho contexto la probabilidad de ocurrencia del símbolo se reduce progresivamente.

Como se plantea en el capítulo siguiente, los modelos de contexto finito obtienen una buena efectividad en la representación det texto, considerando que la probabilidad de qué una letra sea el siguiente símbolo en el mensaje depende fuertemente del conjunto de letras que han aparecido previamente. Por ejemplo, la probabilidad de que después de una q aparezca una u, en un texto en castellano, es mucho más alta que la de cualquier otra letra.

Definición 2.3 (Orden del modelo) *El orden de un modelo de contexto finito es un valor entero que indica la longitud del contexto utilizado en el cálculo de la probabilidad de aparición de cada símbolo considerado por el modelo.*

La construcción de un modelo de contexto finito se lleva a cabo de acuerdo a un orden determinado. Por ejemplo, un modelo de bigramas utiliza únicamente el símbolo previo como contexto para el cálculo de la probabilidad del siguiente. Por lo tanto, éste es un modelo de *orden 1*. Un modelo de trigramas obtiene la probabilidad de un símbolo sobre un contexto formado por los dos símbolos precedentes (*orden 2*) y así sucesivamente para valores de orden crecientes. Un caso "especial" lo representan aquellos modelos en los que no se utiliza ningún contexto para el cálculo de la probabilidad del símbolo siguiente. Éstos se desarrollan sobre la asunción de que la probabilidad de ocurrencia de un símbolo es independiente de los anteriores. Estos modelos se denominan de *orden 0*. La utilización de modelos de orden superior a 0 obtiene una mejor representación de las propiedades del mensaje sólo si el contexto se puede considerar como una secuencia de Markov. Si esto sucede, los datos de entrada son *sensibles al contexto*.

El cálculo de la probabilidad de un símbolo considera, de forma genérica, el número de ocurrencias que éste presenta. Sin embargo, al utilizar un modelo de contexto finito también se necesita almacenar información acerca del contexto en el que dicho símbolo aparece. Esto supone que, para cada contexto, se necesite almacenar la probabilidad de cada uno de los símbolos que aparecen en él. Esta información posibilita la consecución de unas mejores representaciones de las propiedades del mensaje. Dicha distribución se caracteriza como *sesgada* frente a la distribución *plana* que basada en la consideración de equiprobabilidad de cada símbolo. El objetivo principal de un modelo es obtener una distribución lo más sesgada posible de los símbolos del mensaje, de manera que se proporcione una alta probabilidad a aquellos símbolos que aparecen más frecuentemente en un determinado espacio, de tal forma que éstos puedan ser codificados con un menor número de bits.

El cálculo de la probabilidad de un símbolo se puede llevar a cabo sobre modelos de orden diferente. Una vez se dispone del valor obtenido en los modelos que representan cada orden, se precisa un mecanismo de mezclado (*blending*) [BCW90] que permita obtener una probabilidad única a partir de cada una de las obtenidas en los respectivos modelos. Las probabilidades obtenidas en modelos de orden mayor tienden a ofrecer una caracterización más realista de las propiedades de cada símbolo en el mensaje, lo cual conlleva la posibilidad de alcanzar una mejores tasas de compresión.

2.4.1. Tasa de entropía

La tasa de *entropía* [Sha51] de una fuente plantea una medida de la *información media* que ésta proporciona. Este valor depende, exclusivamente, de su naturaleza estadística. En el contexto de

Carácter	% Prob.	Digrama	% Prob.	Trigrama	% Prob.	Tetragrama	% Prob.
·	17,41	e·	3,05	·th	1,62	·the	1,25
e	9,76	·t	2,40	the	1,36	the·	1,04
t	7,01	th	2,03	he·	1,32	·of·	0,60
a	6,15	he	1,97	·of	0,63	and·	0,48
o	5,90	·a	1,75	of·	0,60	·and	0,46
i	5,51	s·	1,75	ed·	0,60	·to·	0,42
n	5,50	d·	1,56	·an	0,59	ing·	0,40
s	4,97	in	1,44	nd·	0,57	·in·	0,32
r	4,74	t·	1,38	and	0,55	tion	0,29
h	4,15	n·	1,28	·in	0,51	n·th	0,23
l	3,19	er	1,26	ing	0,50	f·th	0.21
d	3,05	an	1,18	·to	0,50	of·t	0,21
c	2,30	·o	1,14	to·	0,46	hat·	0,20
u	2,10	re	1,10	ng·	0,44	·tha	0,20
m	1,87	on	1,00	er·	0,39	.···	0.20
f	1,76	·s	0,99	in·	0,38	his·	0,19
p	1,50	,·	0,96	is·	0,37	·for	0,19
g	1,47	·l	0,93	ion	0,36	ion·	0,18
w	1,38	·w	0,92	·a·	0,36	that	0,17
y	1,33	at	0,87	on·	0,35	·was	0,17
b	1,10	en	0,86	as·	0,33	d·th	0.16
,	0,98	ro	0,83	·co	0,32	·is·	0,16
.	0,83	y·	0,82	re·	0,32	was·	0,16
v	0,77	nd	0,81	at·	0,31	t·th	0,16
k	0,49	.·	0,81	ent	0,30	atio	0,15
T	0,30	·h	0,78	e·t	0,30	·The	0,15
”	0,29	ed	0,77	tio	0,29	e·th	0,15
...	...	...	...	...	...	...	...
$\|\mathcal{A}\|$	94	$\|\mathcal{A}\|$	3410	$\|\mathcal{A}\|$	30249	$\|\mathcal{A}\|$	131517
$\mathcal{H}_0$	4,47	$\mathcal{H}_1$	3,59	$\mathcal{H}_2$	2,92	$\mathcal{H}_3$	2,33

Tabla 2.1: Estadísticas del corpus Brown [KF67] (entropías $\mathcal{H}_k$ en bits/símbolo).

la teoría de la información, el valor de la entropía se expresa en *bits/símbolo*. La entropía otorga una cota inferior para la longitud media mínima de palabra de código que un modelo puede conseguir en la representación de un mensaje emitido por la fuente. Esto supone que ningún compresor puede obtener un longitud media menor al valor teórico indicado por la entropía. Sin embargo, el uso de la entropía como cota inferior para la compresión de una fuente es una aproximación muy pobre dado que no tiene en consideración otros factores, asociados con el modelo, como la representación de su alfabeto fuente, la codificación de las estadísticas y los contextos utilizados, etc.

La entropía plantea una interpretación intuitiva de una fuente de datos. Por un lado, si la secuencia de símbolos que forman un mensaje se puede predecir con facilidad, el número de bits necesario para su codificación será significativamente menor al requerido para codificar otra secuencia poco predecible. En el primer caso el valor de $\mathcal{H}$ es más pequeño que en el segundo. Este hecho contrasta, de nuevo, lo anteriormente planteado acerca de que la entropía de una fuente depende de su caracterización estadística y, análogamente, del modelo utilizado para su representación. Esto supone que el valor de la entropía de una fuente depende del orden utilizado en el modelo; por lo tanto, el orden del modelo es un parámetro que debe ser considerado en el cálculo de la entropía.

A continuación se presentan las descripciones genéricas de modelos de orden k a partir de las cuales se deduce la entropía de la fuente. A lo largo de toda esta explicación se considera, como ejemplo real, la experiencia mostrada por Witten y Bell [WB90] en su estudio sobre modelos de representación para una fuente de información que emite mensajes en inglés[3]. Para este tipo de mensajes se acepta la utilización de un *alfabeto fuente* ($\mathcal{A}$) compuesto por 94 caracteres diferentes.

[3]Todos los ejemplos mostrados corresponden con los originalmente publicados en el trabajo de Witten y Bell.

Supongamos que se utiliza un modelo basado en caracteres para la representación de dicha fuente emisora de mensajes en inglés. La correlación existente entre caracteres sucesivos se puede estimar mediante la frecuencia de aparición de secuencias de caracteres de una determinada longitud. Los pares de caracteres consecutivos se refieren como *bigramas*, los triples como *trigramas* y así sucesivamente. Muchos secuencias de caracteres nunca aparecen en los mensajes emitidos por una fuente. Por ejemplo, únicamente el 39 % de los posibles bigramas que pueden formarse a partir de $\mathcal{A}$ se utilizan realmente en los mensajes emitidos por la fuente. Estos datos se aproximan mediante la utilización de una colección real de texto en inglés. El corpus *Brown* [KF67] está compuesto por un conjunto de 500 muestras independientes extraídas de diferentes publicaciones correspondientes al año 1961. Cada una de las muestras está formada por 2000 palabras y, en su conjunto, este corpus plantea una buena representación de textos en inglés considerando la variedad de estilos y autores, así como la temática de cada uno de los textos. La tabla 2.1 presenta una descripción estadística de este corpus (el carácter '·' representa el espacio en blanco) donde puede observarse como sólo el 39 % de los posibles bigramas, que pueden ser formados a partir de $\mathcal{A}$, se utilizan realmente en los mensajes emitidos por la fuente. Este efecto es aún más marcado para secuencias de mayor tamaño, donde sólo el $3,6\,\%$ de los posibles trigramas y el $0,2\,\%$ de los tetragramas se utilizan en los mensajes de la fuente.

Supongamos un alfabeto genérico $\mathcal{A}$ formado por σ símbolos diferentes. Para el ejemplo actual, se asume un valor $\sigma = 94$ (dado que es el número que es el número efectivo de caracteres utilizados en un texto en inglés). Sobre este contexto, se concretan las siguientes definiciones relativas al orden de un modelo.

Modelo equiprobable: los símbolos son estadísticamente independientes los unos de los otros y cada símbolo de $\mathcal{A}$ tiene la misma probabilidad de aparecer. En este caso la tasa de entropía se calcula como

$$\mathcal{H} = \log_2 \sigma \tag{2.2}$$

Este tipo de modelos se ajustan a fuentes de información que emiten mensajes de naturaleza aleatoria en el que la ocurrencia de cada símbolo es independiente de cualquier posible secuencia que lo preceda. Estos modelos, también referidos como modelos de orden -1 [WB90], están basados en una distribución de frecuencia uniforme. Retomando el ejemplo anterior, la fuente modelada emitiría cualquier símbolo del alfabeto con probabilidad 1/94, lo que supone que la fuente representada tendría una entropía $\mathcal{H} = \log_2 94 = 6,55$ bits/símbolo. El texto siguiente representa un posible mensaje construido con el modelo equiprobable. Puede observarse como es una secuencia completamente aleatoria y sin ningún parecido con un texto en inglés salvo que está formado con símbolos procedentes del mismo alfabeto de entrada.

```
)'unHijz'YNvzweOSX,kjJRtyO'$(8)a"#\Dv*;-'';^_o.&uxPI)J'XRfvtOuHIXegO)xZE&vze'*&
w#V[,;<(#v7Nm 1' x/ir$Ix6Ex8O O1plyGDyOa+!/3zAs[U?EH]([sMo,{nXiy_} A>2*->F.RBi'
?9\!wd]&2M3IV&Mk eG>2R<Q2e>Ti8k)SHEeH<kt\$9>[@&aZk(29ti(OC\9uc]cF‘‘ImZ5b^O;T*B5
dH?wa3{I;L^3UlwSW4bFnw(NGD~k 8QcWe_a\F@*'t;Xlr(+8v>\E-:bk;zW9IUx,Oth05rpE.d(<IN
U}kL&gA,>VeW]SJ$'"m2Oz? oE>xaEGQ\};Tevz#gxtEL_JNZR{jgU[,m(75Zt}rLIXCgu+'jj,JOu;
,*$ae0nn9A.P)!{+sZ
```

Modelo de orden 0**:** los símbolos son estadísticamente independientes. Sea p_i la probabilidad de aparición del i-ésimo símbolo del alfabeto. La tasa de entropía viene dada por la ecuación

$$\mathcal{H} = -\sum_{i=1}^{\sigma} p_i \log_2 p_i \tag{2.3}$$

Un modelo de orden 0 representa cada símbolo del alfabeto fuente de acuerdo a su probabilidad particular, independiente de la del resto de símbolos. En el ejemplo tratado sobre el texto inglés, este tipo de modelo representaría una fuente que emitiese caracteres de acuerdo a su probabilidad de aparición (por ejemplo, en la tabla 2.1, puede observarse que la frecuencia del carácter *e* es de 0,0976 o la de *m* que es 0,0187). El siguiente texto es un ejemplo representativo del tipo de mensaje "esperado" para este tipo de modelos. Aunque sigue sin transmitir una idea de texto en inglés, la proporción de los caracteres ya se ajusta a la distribución de probabilidad estimada, aunque no representa correlación alguna (salvo la puramente aleatoria) entre caracteres consecutivos. La entropía se reduce respecto al anterior: $\mathcal{H} = 4,47$ bits/símbolo.

```
fsn'iaad ir lntns hynd,.aais oayimh t n ,at oeote fheotyi t afrtgt oidtsO, wrr
thraeoerdaFr ce.g psNo is.emahntawe,ei t etaodgdna- & em r n nd fih an f tpteaa
lnmas ssn t''bar o be um oon tsrcs et mi ithyoitt h u ans w vsgr tn heaacrY.d e
rfdut y e, a,m<hra Pieodn nyeSrsoto oea nlorseo j r s t w ge g E ikdeAJ .1 eeTJ
iahednn,ngaosl dshoHo eh seelm G os threen nrfigeo,edsoht tgt n tiI a issnin''a
bi''h nht.ebs co efhetntoilgevtnnadrtssaa ka dfnssivb kuniseeao M41 h acdchnr o
noal ic aIhehtr web Yolo aere mblefeuom eomtlkIo h oattogodrinl aw Blbe.
```

Modelo de orden 1: Sea $P_{j|i}$ la probabilidad condicional de que el j-ésimo símbolo del alfabeto aparezca precedido del i-ésimo. La tasa de entropía se obtiene como

$$\mathcal{H} = -\sum_{i=1}^{\sigma} p_i \sum_{j=1}^{\sigma} P_{j|i} \log_2 P_{j|i} \tag{2.4}$$

Este modelo ya considera la correlación existente entre un carácter y su predecesor. Su representación estadística se basa en la frecuencia de aparición de los bigramas de tal forma que cada símbolo se codifica de acuerdo al contexto que representa su predecesor. Aunque este nivel de condicionamiento es aún insuficiente para fuentes de información como la estudiada en el ejemplo actual (se considera que una palabra en inglés tiene una longitud media de, aproximadamente, 5 símbolos [BCW90]), la secuencia mostrada a continuación ya muestra una mayor semejanza con un texto en inglés. Por ejemplo, algunos bigramas como '`e`·', '·`t`' o '`th`' ya aparecen en el fragmento. La entropía de orden 1 se reduce frente a la obtenida en el modelo de orden inferior alcanzando un valor de $\mathcal{H} = 2,92$ bits/símbolo.

```
ne h. Evesicusemes Joul itho antes aceravadimpacalagimoffie ff tineng ads,bathe
nleredisineally, casere o angeryou t manthed t igaroote Bandonede chededienthed
th Bybvey wne, bexpmue ire gontt angig, ay a dy fr t is auld as itresstyTh mery,
winure E thontobe tme geepindus hifethicthed, outed julor hely Lore tothat
batous hthanotonym, thort teler) ILosst aithequther, theero of s s CotPachoucer
he ctevee ange, te athawh tis ld aistevit me athe prube thethicalkehoupalereshe
-nubeascedwhranung of HEammes ani he, d fed olincashed an,
```

Modelo de orden 2: Sea $P_{k|j,i}$ la probabilidad condicional de que el k-ésimo símbolo del alfabeto aparezca precedido j-ésimo y éste, a su vez, esté precedido por el símbolo i-ésimo del alfabeto. En este caso, la tasa de entropía es

$$\mathcal{H} = -\sum_{i=1}^{\sigma} p_i \sum_{j=1}^{\sigma} P_{j|i} \sum_{k=1}^{\sigma} P_{k|j,i} \log_2 P_{k|j,i} \tag{2.5}$$

El presente modelo de orden 2 plantea una evolución del anterior dado que representa la fuente a partir de la frecuencia de sus trigramas. Esta decisión permite estimar la probabilidad de cada símbolo utilizando los dos símbolos precedentes. El texto siguiente representa un mensaje emitido por la fuente tratada en el modelo actual. En él se pueden encontrar trigramas procedentes del idioma inglés como '·th', 'the' o '·he', entre otros. Sin embargo, sigue sin ser un texto inglés "válido" lo que supone que el orden del modelo actual sigue siendo insuficiente para la representación de la fuente estudiada. La entropía actual es $\mathcal{H} = 2,53$ bits/símbolo.

```
he ind wory. Latin, und pow''. I hinced Newhe nit hiske by re atious opeculboui
ly‘‘Whend-bacilling ity and he int wousliner th anicur id ent exon on the 2:26h,
Jusion-blikee thes. I give hies mobione hat not mobot cat In he dis gir achn's
sh.Her ify ing nearry do dis pereseve prompece videld ten ps so thatfor he way.
Inhasiverithe ont thering ing trive forld able nail, 1959 pillaniving boto he
bureofament dectivighe fect who witing me Secitsdshime atimpt the suppecturili
quest.‘‘Henturnsliens he Durvire andifted of skinged mort. Anday hing to de ned
wasucleem ity,
```

Modelo general: Supongamos que B_m representa los primeros m caracteres del texto. La tasa de entropía, en el caso general, viene dada por la siguiente expresión

$$\mathcal{H} = \lim_{m\to\infty} -\frac{1}{j} \sum p\left(B_m\right) \log_2 p\left(B_k\right) \tag{2.6}$$

Este tipo de modelo representa la generalización de todos los casos anteriores. El valor de m, para el caso actual, representa la posibilidad de construir un modelo basado en las estadísticas de secuencias de símbolos de tamaño ilimitado. Sin embargo, este cálculo teórico se convierte en un problema impracticable en un entorno real. En términos prácticos se utiliza un umbral k como orden del modelo. Este valor determina la longitud de las secuencias de símbolos utilizadas en la construcción del modelo de tal forma que la probabilidad, de cada símbolo en el mismo, se estima utilizando como contexto los $k-1$ símbolos anteriores. Los siguientes textos muestran ejemplos de mensajes obtenidos sobre modelos de orden 5 ($\mathcal{H} = 2,61$ bits/símbolo) y de orden 11 ($\mathcal{H} = 0,36$ bits/símbolo). Puede observarse una mejora progresiva en la calidad del resultado obtenido aunque ni siquiera en el último fragmento se está obteniendo un texto completamente correcto. Aunque para $k = 11$ el modelo ya es capaz de identificar numerosas correlaciones entre palabras consecutivas (asumiendo una longitud media de palabra de 5 caracteres), la alta varianza existente en la longitud de las palabras impide detectar todas las correlaciones y esto se traduce en los defectos que pueden ser encontrados en el último fragmento presentado.

```
number diness, and it also light of still try and among Presidential discussion
is department-transcended ‘‘at they maker and for liquor in an impudents to
each chemistry is that American denying it did not feel I mustached through to
the budget, son which the fragment on optically should not even work before
that he was ridiculous little black-body involved the workable of write: "The
Lord Steak a line (on 5 cubic century. When the bleaches suggest connection,
and they were that, but you''. The route whatever second left Americans will
done a m the cold,
```

```
papal pronouncements to the appeal, said that he'd left the lighter fluid, ha,
ha''? asked the same number of temptation to the word 'violent'. ``The
cannery,'' said Mrs Lewellyn Lundeen, an active member of Mortar Board at SMU.
Her husband, who is the Michelangelo could not quite come to be taxed, or for a
married could enroll in the mornings, I was informed. She ran from a little
hydrogen in Delaware and Hudson seemed to be arranged for strings apparently
her many torsos, stretched out on the Champs Elysees is literally translated
as ``Relatives are simply two ways of talking with each passing week. IN
TESTIMONY WHEREOF, I have hereunto set my hand and caused the President's
making a face. ``What's he doing here''? "This afternoon. When he turns upon
the pleader by state law.
```

Todas las definiciones previas sobre el concepto de entropía son consistentes entre sí. En el ámbito de la compresión de datos se trabaja, de forma genérica, con modelos de orden 0, por lo que al citar el concepto de entropía se hace referencia (salvo que se diga lo contrario) a la siguiente definición.

Definición 2.4 (Tasa de entropía de la fuente) *Sea el conjunto de σ sucesos independientes $\mathcal{A}$ con sus correspondientes probabilidades de aparición $\mathcal{P}$. Se define la entropía de la fuente, $\mathcal{H}(\mathcal{P})$, de la siguiente manera*

$$\mathcal{H}(p_1, \ldots, p_\sigma) = \mathcal{H}(\mathcal{P}) = -\sum_{i=1}^{\sigma} p_i \log_2 p_i \tag{2.7}$$

2.4.2. Características de la entropía

La función de entropía, $\mathcal{H}$, se caracteriza por un conjunto de propiedades particulares. Una descripción completa de las mismas se puede encontrar en [Dro02]. A continuación se plantea una revisión resumida de estas propiedades con el fin de mostrar las características esenciales de la entropía.

1. La función $\mathcal{H}$ es *continua* en el intervalo $[0, 1]$. Esto supone que pequeñas variaciones en la probabilidad están sujetas a pequeñas variaciones en la cantidad de información.

2. La función $\mathcal{H}$ es *simétrica*.

3. La función $\mathcal{H}$ tiene *límite superior e inferior*:

$$0 = \mathcal{H}(1, 0, \ldots, 0) \leq \mathcal{H}(p_1, \ldots, p_\sigma) \leq \mathcal{H}\left(\frac{1}{\sigma}, \ldots, \frac{1}{\sigma}\right) = \log_2 \sigma$$

4. *Axioma del grupo*. Si en el conjunto $\mathcal{A} = \{x_1, \ldots, x_\sigma\}$ se forma un grupo $\mathcal{A}_i = \{x_1, \ldots, x_i\}$ entonces, la cantidad de información no cambia después de hacer la escisión y es igual a la información asociada al conjunto $\mathcal{A} - \mathcal{A}_i$ más la información proporcionada por el conjunto $\mathcal{A}_i$, es decir:

$$\begin{aligned}\mathcal{H}(p_1, \ldots, p_i, p_{i+1}, \ldots, p_\sigma) = {} & \mathcal{H}(p_1 + \ldots + p_i, p_{i+1}, \ldots, p_\sigma) \\ & + (p_1 + \ldots + p_i)\mathcal{H}\left(\frac{p_1}{p_1 + \ldots + p_i}, \ldots, \frac{p_i}{p_1 + \ldots + p_i}\right)\end{aligned}$$

5. *Propiedad del conjunto vacío*. El valor de la entropía no se modifica cuando un nuevo evento, con probabilidad cero, es añadido al conjunto de eventos.

$$\mathcal{H}(p_1, \ldots, p_\sigma, 0) = \mathcal{H}(p_1, \ldots, p_\sigma)$$

6. La función $f(\sigma) = \mathcal{H}\left(\frac{1}{\sigma}, \ldots, \frac{1}{\sigma}\right)$ es *monótona creciente*

$$f(n) < f(\sigma + i) \quad \forall\ i > 0\ \wedge\ n > 0$$

por lo que la cantidad de información es directamente proporcional al número de eventos equiprobables.

2.4.3. Eficiencia y redundancia de una fuente

Dada una fuente de información con un alfabeto fuente $\mathcal{A}$ (formado por σ símbolos diferentes) y una distribución de probabilidad $\mathcal{P}$, su entropía $\mathcal{H}(\mathcal{P})$ se maximiza si todos los símbolos del alfabeto son equiprobables:

$$p_i = \frac{1}{\sigma} \quad \forall\ a_i \in \mathcal{A}$$

y su valor es:

$$\mathcal{H}(\mathcal{P}) = \log_2 \sigma$$

Definición 2.5 (Eficacia de la fuente) *Se define la* eficacia de la fuente *o* entropía relativa*, $E(\mathcal{A})$, como la razón entre la entropía real y la entropía máxima de la fuente*

$$E(\mathcal{A}) = \frac{\mathcal{H}(\mathcal{P})}{\log_2 \sigma} \tag{2.8}$$

Definición 2.6 (Redundancia de la fuente) *La* redundancia de la fuente *se calcula mediante la siguiente expresión*

$$R(\mathcal{A}) = 1 - E(\mathcal{A}) \tag{2.9}$$

Nótese que se cumple que $0 < E(\mathcal{A}) \leq 1$, y por lo tanto también $0 < R(\mathcal{A}) \leq 1$. El concepto de redundancia está ligado al exceso de terminología utilizada por la fuente en la generación de mensajes que contienen una cierta cantidad de información. Si una fuente es más redundante que otra dada sus mensajes utilizarán un mayor números de símbolo para expresar la misma cantidad de información.

2.5 Codificación de una fuente de información

La presente sección enfoca la segunda etapa de toda técnica de compresión: la *codificación* del mensaje a partir de su representación en el modelo. A continuación se plantea, de forma general, el contexto en el que se lleva a cabo la etapa de codificación dentro del ámbito de la compresión de datos, enunciando y comentando algunas propiedades esenciales de dicho contexto y de los códigos considerados. En [Dro02] puede encontrarse una explicación completa y detallada de los teoremas relacionados y sus correspondientes demostraciones y pruebas.

Anteriormente se comentaba que el canal de comunicación considerado en un proceso de compresión de datos se puede caracterizar como un canal sin ruido ni memoria. Esto supone que la codificación puede ser, igualmente, referida como *codificación sin ruido ni memoria.* Consideremos un alfabeto fuente $\mathcal{A} = \{x_1, \ldots, x_\sigma\}$ formado por σ símbolos diferentes caracterizados por la siguiente distribución de probabilidad: $\mathcal{P} = \{p_1, \ldots, p_\sigma\}$. Estos símbolos se codifican mediante las palabras de código $\mathcal{C} = \{c_1, \ldots, c_\sigma\}$ con longitudes $|\mathcal{C}| = \{|c_1|, \ldots, |c_\sigma|\}$. De esta manera, la codificación de un símbolo $x_i \in \mathcal{A}$, con probabilidad de aparición p_i, se lleva a cabo mediante la palabra de código c_i cuya longitud es de $|c_i|$ unidades.

Un *código* se define, de forma sencilla, como una asociación biunívoca entre los conjuntos $\mathcal{A}$ y $\mathcal{C}$ de tal forma que cada símbolo x_i está asociado con una y sólo una palabra de código c_i. Por lo tanto, la *codificación* es una aplicación inyectiva definida como:

$$\psi : \mathcal{A} \to \mathcal{C}$$

De forma análoga, la *decodificación* se define como la aplicación inyectiva inversa de la anterior:

$$\psi^{-1} : \mathcal{C} \to \mathcal{A}$$

Un código cuyas palabras están formadas por bits se refiere como *código binario*. El interés principal de la compresión de datos es minimizar la *longitud media esperada* de las palabras de código $\mathcal{C}$ [MT02] bajo la consideración de obtener la representación más compacta posible de un determinado mensaje. El concepto de longitud media esperada se define como:

$$\bar{L}(\mathcal{C}, \mathcal{P}) = \frac{\sum_{i=1}^{\sigma} p_i |c_i|}{\sum_{i=1}^{\sigma} p_i} \tag{2.10}$$

Además, para una determinada distribución de probabilidad $\mathcal{P}$, todos los códigos $\mathcal{C}$ cumplen la siguiente desigualdad [BCW90]:

$$\mathcal{H}(\mathcal{P}) \leq \bar{L}(\mathcal{C}, \mathcal{P})$$

Definición 2.7 (Código de longitud variable) *Un código $\mathcal{C}$, con σ palabras de código, es de longitud variable si, al menos, dos de esas palabras cumplen lo siguiente:*

$$\exists \quad 1 \leq i \leq \sigma \ \wedge \ 1 \leq j \leq \sigma \ / \ |c_i| \neq |c_j|$$

Definición 2.8 (Código descifrable) *Un código $\mathcal{C}$ es* descifrable *si sólo existe una manera de dividir la secuencia de palabras de código $c_{i_1} c_{i_2} \ldots c_{i_k}$ en palabras de código de forma inequívoca. Es decir, si $c_{i_1} c_{i_2} \ldots c_{i_k} = c_{j_1} c_{j_2} \ldots c_{j_k}$ entonces $\forall s, i_s = j_s$ (y por lo tanto $c_{i_s} = c_{j_s}$).*

Definición 2.9 (Propiedad del prefijo) *Un código $\mathcal{C}$ posee la propiedad del prefijo si ninguna palabra de código, c_i, se puede obtener a partir de la concatenación de sendas palabras $c_j c_k$. Enunciado de forma equivalente, un código posee esta propiedad si ninguna de sus palabras es prefijo de otra.*

Definición 2.10 (Código libre de prefijo) *Un código $\mathcal{C}$ es libre de prefijo (o instantáneo) si posee la propiedad del prefijo.*

Los códigos libres de prefijo son un subconjunto particular de todos los códigos descifrables. La propiedad del prefijo garantiza la decodificación progresiva de cualquier secuencia de bits obtenida sobre la aplicación de estos códigos dado que ninguna palabra puede ser prefijo de otra y, por tanto, puede ser decodificada sin ambigüedad. De la misma forma, la propiedad del prefijo evita la necesidad de utilizar símbolos de separación entre diferentes palabras de código.

Definición 2.11 (Código de redundancia mínima) *Un código $\mathcal{C}$ es de redundancia mínima (u óptimo), para el conjunto de probabilidades $\mathcal{P}$, si $\bar{L}(\mathcal{C}, \mathcal{P}) < \bar{L}(\mathcal{C}', \mathcal{P})$ para cada código $\mathcal{C}'$ libre de prefijo de σ símbolos.*

En otras palabras, un código es de redundancia mínima, para una distribución de probabilidad dada, si no existe ningún otro código libre de prefijo cuya longitud media esperada sea menor que la obtenida por el código considerado. En una degeneración de la terminología, estos códigos libres de prefijo y de redundancia mínima suelen ser referidos como códigos de Huffman en alusión al famoso algoritmo de Huffman [Huf52] del cual se obtienen códigos con estas propiedades. Sin embargo, estrictamente hablando, los códigos de Huffman son sólo un subconjunto de todos los posibles códigos libres de prefijo y de redundancia mínima que pueden ser obtenidos.

Los códigos de redundancia mínima fueron masivamente utilizados hasta finales de la década de los 70. En este momento irrumpen nuevos paradigmas de compresión de la mano de los trabajos de Ziv y Lempel [ZL77, ZL78] que mejoran los códigos de redundancia mínima en diferentes aspectos. Sin embargo, dichos códigos de redundancia mínima siguen siendo una opción competitiva en diferentes tipos de aplicaciones como, por ejemplo, los relacionados con el ámbito de la recuperación de información.

Corolario 2.1 (Código compresor) *Para que un código $\mathcal{C}$ pueda ser un código compresor, $\mathcal{C}$ debe ser descifrable y de longitud variable. Además es conveniente que sea un código libre de prefijo y de redundancia mínima.*

$\mathcal{A}$	p_i	Código 1	Código 2	Código 3
x_1	0,67	000	00	0
x_2	0,11	001	01	100
x_3	0,07	010	100	101
x_4	0,06	011	101	110
x_5	0,05	100	110	1110
x_6	0,04	101	111	1111
$\bar{L}(\mathcal{C},\mathcal{P})$		3,00	2,22	1,75

Tabla 2.2: Ejemplo de tres códigos libres de prefijo con su longitud media esperada, $\bar{L}(\mathcal{C},\mathcal{P})$, expresada en bits por símbolo (extraído de [MT02]).

Este corolario se entiende fácilmente sobre el siguiente ejemplo. Consideremos el alfabeto $\mathcal{A} = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ ($\sigma = 6$) presentado en la tabla 2.2. Las probabilidades p_i asociadas a cada símbolo se presentan en la segunda columna. La codificación se lleva a cabo con un código binario para el que se consideran las tres alternativas mostradas en las tres columnas siguientes de la tabla.

- El *código 1* se construye sobre una representación binaria estándar que requiere $\lceil \log_2 \sigma \rceil = 3$ bits para cada palabra de código. Éste no es un código *completo* dado que no utiliza todos sus posibles prefijos (puede observarse que ninguna palabra de código empieza por `11`).

- El *código 2* plantea un código completo formado a partir del anterior. Los símbolos más frecuentes pasan a estar codificados con palabras de 2 bits gracias a la utilización del prefijo no considerado en el código anterior. A pesar de esta última decisión, el presente código sigue manteniendo la propiedad del prefijo. La asignación de códigos considerada, en el caso actual, deja claro uno de los principios fundamentales de la compresión: representar con palabras de código más cortas los símbolos más frecuentes reduce la longitud media esperada y con ello el tamaño final del mensaje comprimido. Puede observare como el valor $\bar{L}(\mathcal{C},\mathcal{P})$ para el código 2 es de 2,22 bits/símbolo frente a los 3 bits/símbolo obtenidos por el código 1.

- El *código 3* también es completo. Su diferencia con el anterior radica en un mejor aprovechamiento de las palabras de código disponibles promocionando la codificación del símbolo más frecuente con un único bit. Este código obtiene el mejor valor de $\bar{L}(\mathcal{C},\mathcal{P})$ (1,75 bits/símbolo), postulándose como un código de reduncia mínima para el alfabeto actual y la distribución de probabilidad que lo caracteriza en la fuente emisora.

La entropía obtenida (de acuerdo a la ecuación 2.7) para la distribución de probabilidad anterior es de 1,65 bits/símbolo. La eficiencia de un código se estima sobre la proximidad de su longitud media esperada respecto al valor teórico establecido por la entropía. Esta relación se puede medir, porcentualmente, mediante el concepto de *ineficiencia* relativa de un código respecto a la entropía:

$$100 \times \left(\frac{\bar{L}(\mathcal{C},\mathcal{P})}{\mathcal{H}(\mathcal{P})} - 1 \right) \tag{2.11}$$

De acuerdo con este concepto, los códigos anteriores son un 82 %, 35 % y 6 % ineficientes respecto a la entropía de la fuente para la distribución de probabilidad considerada.

2.5.1. La desigualdad de Kraft

Dada una fuente de información caracterizada sobre un alfabeto $\mathcal{A}$ y una distribución de probabilidad $\mathcal{P}$, una de las cuestiones básicas que se formulan en el ámbito de la compresión radica en conocer el límite inferior de la longitud media esperada para un código descifrable construido para dicha fuente. Supongamos que cada símbolo $x_i \in \mathcal{A}$ tiene una probabilidad $p_i = 2^{-k_i}$, de tal forma que si este valor es sustituido en la ecuación 2.1 se obtiene $I(x_i) = k_i \ / \ k_i \in \mathbb{Z}$. La generación de la palabras de códigos asociadas a los símbolos de la fuente ($|c_i| = k_i$) permite obtener un código cuyo longitud esperada es mínima y coincidente con el valor de la entropía. Esta misma observación representa la base sobre la que se describe el siguiente teorema [Kra49]:

Teorema 2.1 (Teorema de Kraft) *Un código binario libre de prefijo, formado por el conjunto de palabras de código* $\{c_1, \ldots, c_\sigma\}$*, debe cumplir la siguiente desigualdad*

$$\sum_{i=1}^{\sigma} 2^{-|c_i|} \leq 1$$

El Teorema de Kraft garantiza encontrar un código libre de prefijo para un conjunto de longitudes determinado que cumple la desigualdad anterior. Por otro lado, un código puede satisfacer este teorema y no ser un código libre de prefijo. Esta relación puede ser invertida de tal forma que el cumplimiento del Teorema de Kraft es un requisito indispensable para ser un código libre de prefijo. Si la suma de Kraft, $K(C)$, definida como:

$$K(\mathcal{C}) = \sum_{i=1}^{\sigma} 2^{-|c_i|} \tag{2.12}$$

es mayor que 1, entonces el código C no es libre de prefijo. El ejemplo más obvio lo representa un código formado por σ palabras de código c_i de longitud constante $|c_i| = 1$. La suma de Kraft para este código da un resultado de $\sigma/2$, lo que supone que un código libre de prefijo para estas longitudes sólo es posible si $\sigma \leq 2$.

Años después, en 1956, McMillan [McM56] extendió este resultado. Si la suma de Kraft para un código $\mathcal{C}$ es menor o igual a 1 ($K(C) \leq 1$), entonces existe otro código $\mathcal{C}' \ / \ \bar{L}(\mathcal{C}',\mathcal{P}) = \bar{L}(\mathcal{C},\mathcal{P}) \wedge |\mathcal{C}'| = |\mathcal{C}|$ cuyas palabras de código están libres de prefijo. Esto supone que un código $\mathcal{C}$

que satisface el teorema de Kraft puede ser transformado en un nuevo código $\mathcal{C}'$ libre de prefijo con la misma longitud media que el código original.

Retomando los códigos presentados en la tabla 2.2, se obtienen sus sumas de Kraft: $K(C_1) = 0,75$, $K(C_2) = 1$ y $K(C_3) = 1$. Como se planteaba anteriormente el código C_1 no es completo. Sobre la desigualdad de Kraft se puede obtener una caracterización concreta para aquellos códigos que si lo son:

Definición 2.12 (Código completo) *Un código $\mathcal{C}$ es completo si y sólo si es un código libre de prefijo y su suma de Kraft, $K(\mathcal{C})$, es igual a uno.*

2.6 Clasificación de los métodos de compresión

Los métodos de compresión se pueden clasificar atendiendo a varios criterios. Inicialmente, se planteó la división más general que distingue aquellos métodos con pérdida (*lossy*) de los métodos sin pérdida (*lossless*). A continuación se presenta una clasificación más detallada, teniendo en cuenta las propiedades específicas de las diferentes formas en las que el modelado y la codificación de la fuente pueden llevarse a cabo dentro de un esquema de compresión. En un primer nivel, los métodos de compresión se pueden clasificar en *estadísticos* y *basados en diccionario* [BCW90]. A su vez, dentro de cada una de estas familias se pueden obtener diferentes subdivisiones basadas en el *tipo de modelado* llevado a cabo y en las propiedades particulares de sus alfabetos fuente y destino[4].

En primer lugar se analizan los diferentes criterios considerados para la subdivisión de los métodos de compresión. A continuación, se caracterizan de forma general los métodos estadísticos y basados en diccionario atendiendo a las propiedades particulares que éstos presentan sobre los criterios anteriores.

Tipo de modelado. La etapa de modelado se puede llevar a cabo sobre tres tipos independientes de procesos: *estáticos*, *semi-estáticos* y *adaptativos* [BWC89]. La decisión del tipo de modelado tiene una fuerte implicación en la etapa de codificación.

- **Modelado estático:** toma como punto de partida una distribución de probabilidad prefijada y compartida por compresor y descompresor. Esta distribución se construye sobre una visión genérica de la fuente de datos, asumiendo que los mensajes generados por la misma se van a ajustar a las premisas consideradas. Por lo tanto, la distribución de probabilidad es independiente del mensaje, de tal forma que la efectividad del compresor depende fuertemente de la similitud existente entre la distribución considerada y la asociada al mensaje a comprimir. El uso de este tipo de modelos se da en entornos que exigen una alta velocidad de cómputo y sencillez en la operativa. El estándar `JPEG`, para la compresión de imágenes, es uno de sus ejemplos más representativos. En esta aplicación se conoce de antemano la función de distribución esperada para los coeficientes espectrales filtrados.

- **Modelado semi-estático:** a diferencia del anterior, el modelado semi-estático construye un modelo específico para cada mensaje emitido por la fuente de información. Este tipo de modelado se lleva a cabo dentro de un proceso de compresión en dos fases. La primera pasada, sobre el mensaje, obtiene una representación específica de sus propiedades estadísticas. Esta información se utiliza para la construcción del modelo que permanece estático a lo largo de la etapa de compresión, de tal forma que un símbolo fuente siempre se

[4]Las propiedades del alfabeto destino se determinan de acuerdo al código seleccionado para la representación comprimida del mensaje.

representa por la misma palabra de código. Nótese que al usar un modelado semi-estático, el descompresor desconoce el modelo utilizado por el compresor de tal forma que éste debe ser explicitamente suministrado junto al mensaje comprimido. Este tipo de modelado se utiliza, de forma mayoritaria, en entornos relacionados con la recuperación de información dado que el mensaje comprimido puede ser accedido y consultado directamente.

- **Modelado adaptativo:** este tipo de modelado (al igual que el anterior) utiliza un modelo específico para la representación de cada mensaje. Basado en un conjunto de premisas iniciales, el modelado adaptativo construye y actualiza el modelo con cada símbolo fuente que codifica. Esto supone una construcción progresiva del modelo con el procesamiento del mensaje, de forma que cada símbolo se codifica de acuerdo al estado particular del modelo en cada paso del proceso. Esta propiedad supone que un mismo símbolo de entrada se puede representar, en el resultado comprimido, mediante palabras de código diferentes. El modelado adaptativo se desarrolla en una única etapa, facilitando el procesamiento del mensaje como si fuese un flujo de información continuo dado que cada símbolo se codifica al ser recibido y el modelo se actualiza de acuerdo a dicho evento. Este último hecho dota de flexibilidad al modelo en construcción dado que, de acuerdo a su actualización progresiva, es capaz de adaptarse a las diferentes propiedades que pueda presentar el mensaje. Por ejemplo, supongamos un símbolo que aparece muy frecuententemente al principio del mensaje mientras que en el resto aparece sólo de forma ocasional. Un modelado adaptativo utizaría palabras de código de menor longitud para la codificación del símbolo en la parte inicial del mensaje mientras que el resto de ocurrencias se codificarían con una longitud de palabra mayor, dada la pérdida de significatividad del símbolo a lo largo del mensaje. Por su parte, un modelado semi-estático no tendría en cuenta estas propiedades y representaría el símbolo sobre un código prefijado con longitud intermedia. Esto tiende, en una gran mayoría de los casos, a aumentar la longitud media de la palabra de código, reduciendo la efectividad relativa del compresor semi-estático frente al adaptativo. Además, un compresor adaptativo no precisa suministrar al descompresor el modelo utilizado en compresión dado que éste puede ser reconstruido exactamente a partir de las mismas asunciones iniciales consideradas en el compresor. Por otra parte, los modelos adaptativos están sujetos a un proceso de actualización constante que puede suponer una importante sobrecarga de cómputo. Complementariamente, estos modelos se enfrentan a la decisión de cómo tratar cada nuevo símbolo que precise ser añadido al modelo.

La elección de un tipo u otro de modelado no puede establecerse de forma general dado que no existe un criterio global que permita decidir que un tipo de modelo es mejor que los restantes. Esta decisión está complementamente influenciada por el contexto en el que el compresor va a ser utilizado. Una comparación analítica entre los modelos adaptativos y no adaptativos se puede encontrar en [BCW90].

Puede considerarse que los métodos adaptativos son los más utilizados en aplicaciones genéricas dado que permiten realizar la compresión del mensaje en una sola etapa. Además, como se ha comentado previamente, este tipo de modelado no requiere suministrar, explícitamente, al descompresor el modelo utilizado en compresión. La codificación de un símbolo, sobre un modelo adaptativo, evoluciona a lo largo de todo el proceso de compresión lo cual implica que la descompresión requiera una ejecución secuencial desde el principio del mensaje. Esta propiedad impide la descompresión parcial del mensaje a partir de una posición aleatoria en el mismo, imposibilitando su utilización práctica en sistemas de recuperación de información en los que es esencial tanto el acceso aleatorio al mensaje comprimido como la descompresión de un fragmento del mismo. Zobel y Moffat [ZM95] estudiaron la posibilidad de construir un modelado adaptativo por bloques concluyendo que no es una alternativa competitiva para propósitos de

recuperación de información.

Alfabeto fuente. Cómo se comentaba en §2.3, la primera decisión que se debe tomar en la etapa de modelado es la definición de qué es considerado un *símbolo* para el modelo. Esta decisión tiene una repercusión importante en la efectividad de la técnica construida. La clasificación de métodos de compresión, de acuerdo a las propiedades de su alfabeto fuente, depende fuertemente del contexto de aplicación. Por ejemplo, como se plantea en el capítulo siguiente, la compresión de texto puede ser afrontada desde sendas perspectivas orientadas a caracteres, palabras e, incluso, secuencias de palabras (frases). El modelo se construye, de forma específica, atendiendo a las propiedades derivadas de la elección del tipo de símbolo, obteniendo compresores de texto orientados a caracteres, palabras o frases.

Alfabeto destino. La elección de un determinado código, para la representación del mensaje, supone un nuevo criterio de clasificación de los métodos de compresión. Básicamente, este aspecto tiene que ver con la unidad mínima de representación considerada en el código construido. En lo que respecta a los objetivos de este trabajo de tesis, podemos clasificar los compresores en orientados a bit y a byte, cuya principal diferencia radica en la unidad mínima sobre la que se lleva a cabo la construcción del código. Esto es, un código orientado a bit obtiene sus palabras de código como secuencias de bits mientras que el orientado a byte hace lo propio sobre secuencias de bytes. La elección del alfabeto destino tiene una importancia significativa tanto en la efectividad del compresor (la longitud media de un código orientado a byte es mayor que la de un código orientado a bit) como en su eficiencia, considerando que las operaciones a nivel de byte se ejecutan de forma más eficiente que sus equivalentes a nivel de bit.

2.6.1. Métodos estadísticos.

Los métodos estadísticos reflejan, a la perfección, la combinación *modelado+codificación* planteada como base de una técnica de compresión. Salvo que se trate de una técnica estática, la etapa de modelado se centra en la estimación de las estadísticas del mensaje de acuerdo a la definición de símbolo considerada. Para ello considera dos políticas [Say02] relacionadas con el propio orden del modelo. Por un lado se puede calcular la probabilidad de cada símbolo como la ratio de su número de ocurrencias respecto al tamaño total del texto. Este tipo de modelado refleja las estadísticas globales del mensaje y calcula la probabilidad de cada símbolo con independencia del resto de símbolos. Por tanto, se construye como un modelo de orden 0. Por otro lado se considera la construcción de un modelo estadístico de orden superior. Como se plantea a nivel teórico, este tipo de representación estima la probabilidad de cada símbolo utilizando como contexto los k símbolos precedentes.

La etapa de codificación se centra en la asignación de palabras de código de menor longitud para la representación de los símbolos más frecuentes. La política de asignación de estos códigos se lleva a cabo de acuerdo al objetivo de obtener la longitud media de palabra más próxima posible al umbral inferior establecido por la entropía.

2.6.2. Métodos basados en diccionario.

Este tipo de métodos tratan de representar el mensaje original de una forma más compacta a través de la construcción de un diccionario de frases[5]. La efectividad de estos métodos se basa en la efectividad obtenida en el proceso de sustitución de las frases por referencias a sus respectivas

[5]Se entiende por frase, en el contexto actual, cualquier secuencia de símbolos consecutivos almacenada en el diccionario.

entradas en el diccionario. Desde este punto de vista, el diccionario puede ser entendido como una función inyectiva $\mathcal{D} : \mathcal{A} \rightarrow \mathcal{C}$ que mapea cada entrada en el diccionario en una palabra de código. De acuerdo a lo comentado en la sección anterior, respecto al orden del modelo, se entiende que una representación basada en diccionario es de orden 0 dado que la representación de cada frase depende exclusivamente de su localización en el diccionario.

Aunque los métodos basados en diccionario no diferencian, explícitamente, las etapas de modelado y codificación, se puede considerar la existencia de dos fases relativamente independientes [BYRN99]. La primera fase de *construcción del diccionario* se puede llevar a cabo sobre procesos estáticos, semi-estáticos u adaptativos. La segunda fase se centra en el reemplazo de cada símbolo en el mensaje por su referencia en el diccionario. En lo que respecta a la fase de construcción, un método estático parte de una organización prefijada del diccionario. La construcción semi-estática u adaptativa responde a las propiedades genéricas consideradas para cada tipo de proceso. Nótese que para una construcción semi-estática del diccionario se precisa codificar su organización final junto al mensaje comprimido. Este tipo de compresores se utilizan, de forma generalizada, en entornos de recuperación de información. Su efectividad depende fuertemente de la organización del diccionario ya que ésta representa la base para la construcción de la función $\mathcal{D}$ sobre la que se considera un esquema de codificación determinado.

2.7 Efectividad de los métodos de compresión

La bondad de un método de compresión comprende sendos análisis de eficiencia y efectividad. El primero de ellos se centra en la cantidad de espacio en memoria requerido por el compresor junto con el tiempo de cómputo que éste necesita para la ejecución de sus procesos de compresión y descompresión. Por su parte, el análisis de efectividad evalúa la capacidad compresiva de un método a partir del tamaño final que éste obtiene en el proceso de compresión de un determinado mensaje. Supongamos un mensaje original de tamaño u bytes, su resultado comprimido en n bytes y c como el número de bits utilizados para representar un símbolo en el alfabeto original. La efectividad de un método de compresión se mide, sobre estos valores, de alguna de las siguientes maneras:

Tasa de compresión: presenta una tendencia creciente con la mejora de la efectividad del compresor. Este valor se calcula como:

$$\left(1 - \frac{n}{u}\right) \tag{2.13}$$

Número de bits por símbolo (BPS): indica el número de bits utilizados en la representación de un símbolo del mensaje original y se calcula mediante la siguiente fórmula:

$$c \times \frac{n}{u} \tag{2.14}$$

Razón de compresión: mide el tamaño proporcional del mensaje comprimido respecto al tamaño del mensaje original. Se calcula como:

$$100 \times \frac{n}{u} \tag{2.15}$$

Factor de compresión (X:1): indica la proporción existente entre el tamaño del mensaje sin comprimir y el tamaño de su resultado comprimido. Este valor se expresa típicamente de la forma `X:1` donde X es un entero y se calcula mediante la fórmula:

$$X = \frac{u}{n} \tag{2.16}$$

2.8 Limitaciones de la compresión

Es matemáticamente imposible obtener un algoritmo de compresión sin pérdida capaz de reducir en, al menos un bit, *todos* los mensajes de un mismo tamaño. Aún así, a lo largo de la historia se encuentran algunos casos que intentan contradecir este hecho. De forma genérica, estas técnicas justifican su hallazgo sobre un proceso recurrente en n pasos basado en la compresión, en cada uno de ellos, del resultado obtenido en el paso justo anterior.

Actualmente, a través del denominado *Hutter Prize*[6] se trata de medir la compresibilidad del conocimiento humano. Sin embargo, el objetivo de esta propuesta es realista dentro de las características presentadas en esta sección dado que no plantea la construcción de un compresor capaz de comprimir excelentemente cualquier mensaje que contenga conocimiento humano, sino que centran su *benchmark* en un texto único (denominado `enwik8`) obtenido a partir de los 100 millones de caracteres primeros de la versión inglesa de Wikipedia[7]. Los resultados obtenidos son francamente competitivos en lo que respecta a la efectividad de las técnicas (actualmente, y desde el 23 de Mayo de 2009, Alexander Ratushnyak ostenta la mejor marca con un $15,49\,\%$ de ratio), sin embargo son soluciones específicas para el corpus propuesto, lo que limita su capacidad compresiva al contexto de experimentación utilizado. Este hecho demuestra, desde una perspectiva práctica, la limitación teórica de la compresión que se pretende mostrar en esta sección.

Teorema 2.2 (Teorema del recuento) *Ningún algoritmo de compresión sin pérdida puede comprimir* todos *los ficheros de tamaño mayor o igual a n bits, para cualquier entero* $n \geq 0$.

Prueba: Supongamos que un algoritmo sin pérdida puede comprimir todos los mensajes de tamaño mayor o igual a n bits. Esto supone que al comprimir los 2^n mensajes de tamaño n bits, el tamaño de todos los mensajes comprimidos tiene que ser menor o igual a $n-1$ bits. En este mismo tamaño se encuentran $2^n - 1$ diferentes mensajes comprimidos (dado que se obtienen 2^{n-1} mensajes comprimidos de tamaño $n-1$, 2^{n-2} mensajes comprimidos de tamaño $n-2$, y así sucesivamente hasta alcanzar un mensaje comprimido en 0 bits). Si esto fuese cierto se encontrarían, al menos, dos mensajes diferentes cuyo resultado comprimido sería el mismo. Por lo tanto, el algoritmo de compresión no sería sin pérdida. □

La prueba anterior se conoce como *argumento del recuento* y se basa en el conocido como *problema del palomar*. Sus orígenes se remontan al año 1834 cuando Johann Dirichlet sugiere su primera formalización. En términos comunes, este problema plantea que dadas "10 palomas" y "9 agujeros", al menos uno de estos agujeros tiene que estar ocupado por dos de estas palomas. Este problema se puede interpretar, desde la perspectiva de la compresión de datos, como una asociación genérica de dos conjuntos: mensajes originales y mensajes comprimidos. Si un algoritmo de compresión sin pérdida pudiese transformar cada uno de los mensajes originales en un mensaje comprimido diferente, dicho algoritmo sería una asociación biyectiva entre dos conjuntos del mismo tamaño. Sin embargo, como planteaba la prueba anterior, dados 2^n mensajes originales formados por n bits, la compresión de al menos dos de ellos será la misma dado que sólo existen $2^n - 1$ posibilidades de representar dichos mensajes en un tamaño menor al original.

[6] http://prize.hutter1.net/
[7] http://wikipedia.org

Después del tiempo
que he perdido
en aventuras
sin sentido ...

Enrique Urquijo

3

Compresión de Texto

La *compresión de texto* [BCW90] se centra en encontrar formas alternativas de representación que permitan almacenar la misma información en un menor espacio [BYRN99]. Por lo tanto, los compresores de texto se encuadran en la familia de técnicas sin pérdida (*losless*) dado que el mensaje obtenido en la descompresión coincide exactamente con el mensaje original. El fundamento de estas técnicas radica en un modelado del texto que permita identificar las regularidades con las que se construye un mensaje en lenguaje natural y, a partir de ellas, eliminar la redundancia subyacente a la información contenida en el mensaje original. Esto permite reducir el espacio ocupado por el mensaje comprimido, mejorando las necesidades de almacenamiento así como la eficiencia temporal tanto en los procesos de transferencia en disco como en los de transmisión de red. Además, algunos de estos métodos de compresión ofrecen, de forma complementaria, la posibilidad de acceder a la información representada en el texto sin necesidad de llevar a cabo su descompresión previa. Estos procesos de búsqueda de información, sobre el texto comprimido, se pueden llevar a cabo hasta 8 veces más rápidos que los procesos equivalentes sobre el texto original [dMNZBY00].

El coste al que están sujetos estos logros es, básicamente, el tiempo necesario para la compresión y descompresión del texto. Sin embargo, el coste relativo de estos procesos tiende a ser cada vez menos significativo dado el considerable aumento de la velocidad de cómputo de los procesadores respecto a la velocidad de transferencia en disco. Esto conlleva que el ahorro en espacio, por sí mismo, no sea importante en algunos contextos de aplicación considerando la notable reducción del coste de almacenamiento. No obstante, el uso de representaciones compactas del texto permite conseguir mejoras significativas en el rendimiento asociado a la utilización de dispositivos lentos (como discos y canales de comunicación) favoreciendo el uso de la compresión en el manejo de grandes colecciones en entornos de aplicación como la WWW o, más concretamente, en bibliotecas digitales y bases de datos documentales.

El presente capítulo aborda un amplio repaso del área en el que se encuadra la compresión de texto. En primer lugar (§3.1) se comentan algunas propiedades fundamentales del lenguaje natural. Este conocimiento empírico facilita, notablemente, el entendimiento de la estructura de un texto y con ello la identificación de la redundancia almacenada en él. A partir de esta contextualización se inicia un resumen general de las distintas técnicas de compresión sin pérdida aplicables al campo de la compresión de texto. La organización de las secciones subsiguientes se lleva a cabo sobre la consideración dada, en el capítulo anterior, acerca de los métodos de compresión. La Sección §3.2 plantea una revisión genérica de las propiedades de los modelos obtenidos sobre procesos adaptativos y semi-estáticos utilizados, a continuación, en la descripción de compresores estadísticos y basados en diccionario. Posteriormente, la Sección §3.3 explica las principales familias de códigos utilizadas para compresión, tratando con mayor detalle aquellas utilizadas en el desarrollo del presente trabajo de tesis. La Sección §3.4 se centra en los compresores de naturaleza estadística, prestando una atención especial al algoritmo `PPM`. La Sección §3.5 estudia los métodos de compresión basados en diccionario, dentro de los cuales se muestra una visión general de las propuestas basadas en gramáticas libres de contexto. A

pesar de que, clásicamente, estas son las dos grandes familias de compresores, en la Sección §3.6 se tratan los métodos de compresión basados en preprocesamiento. Aunque estas técnicas no plantean una forma de compresión al uso, en los últimos años se han propuesto diferentes compresores basados en transformaciones previas del mensaje con una efectividad competitiva. Esta sección plantea un repaso de las mismas, prestando especial atención a la Transformada de *Burrows-Wheeler* [BW94].

3.1 Conceptos básicos

Los mensajes de texto se construyen sobre el lenguaje natural utilizado por los humanos en sus actos generales de comunicación. La información intercambiada en estos procesos de comunicación es rica y variada aunque también predecible de acuerdo tanto a las reglas sintácticas que rigen la estructura de estos mensajes como a otras propiedades de carácter semántico.

Cómo se planteaba en el capítulo anterior, la primera decisión a tomar a la hora de diseñar una técnica de compresión radica en la determinación de qué es considerado cómo *símbolo*. Para el caso actual, la decisión debe tener en cuenta la perspectiva humana ante la interpretación de un mensaje en lenguaje natural dado que ésta permitirá una mejor identificación de las regularidades subyacentes al mismo. Es un hecho contrastado que una persona no interpreta un texto como una secuencia de caracteres individuales sino como una secuencia de *palabras* y *separadores*. Esto conlleva la necesidad de delimitar claramente qué es una palabra y qué un separador. En el presente trabajo se consideran las siguientes definiciones:

Definición 3.1 (Palabra) *Una palabra es una secuencia maximal de caracteres alfanuméricos.*

Definición 3.2 (Separador) *Un separador es una secuencia maximal de caracteres no alfanuméricos entre dos palabras consecutivas.*

Por lo tanto, un mensaje textual se puede plantear como una secuencia en *alternancia* de palabras y separadores construida de acuerdo a la semántica que éste transmite. Existen diferentes leyes experimentales que nos permiten entender algunas de las regularidades inherentes al lenguaje natural. A continuación se repasan las leyes más significativas en el contexto actual y finalmente se plantean las diferentes alternativas seguidas para el modelado de lenguaje natural orientado a compresión.

Ley de Heaps. La Ley de Heaps [Hea78, BYRN99] aproxima la relación entre el número total de palabras de un texto (n) y el número de palabras diferentes σ (este conjunto de palabras es referido como *vocabulario*). Esta relación se establece como $\sigma = \alpha n^{\beta}$, donde α y β son parámetros obtenidos empíricamente que dependen directamente del mensaje y del idioma subyacente al mismo. Más concretamente, el parámetro β depende de la homogeneidad del texto: valores crecientes de β indican una mayor heterogeneidad en el texto. Para textos en inglés, los valores anteriores se estiman como $10 \leq \alpha \leq 100$ y $0,4 \leq \beta \leq 0,6$. La Ley de Heaps, en conclusión, sugiere un crecimiento sublineal del tamaño del vocabulario respecto al tamaño del texto.

Ley de Zipf. La Ley de Zipf [Zip49, BYRN99] plantea una buena estimación de la distribución de frecuencia que siguen las palabras en un texto en lenguaje natural. Es un hecho conocido que dicha distribución posee una naturaleza muy asimétrica dado que un pequeño conjunto de palabras acumula un alta frecuencia de aparición mientras que las palabras del conjunto restante presentan una frecuencia muy baja. La Ley de Zipf cuantifica esta propiedad como:

$$p_i = \frac{A_x}{i^\theta}$$

donde i representa el ranking de la palabra en el vocabulario ($i = 1 \dots \sigma$), θ es un parámetro dependiente del texto ($1 < \theta < 2$ para textos en inglés) y $A_x = \frac{1}{\sum_{i>0} 1/\theta^i} = 1/\zeta(\theta)$ es un factor de normalización. $\zeta(\theta)$ es conocida como función *Zeta*.

Ley de Mandelbrot-Zipf. Mandelbrot [Man53] plantea una modificación sobre la Ley de Zipf, añadiendo un nuevo parámetro (C) dependiente del texto. La Ley de Mandelbrot-Zipf calcula la probabilidad del i-ésimo símbolo más frecuente como:

$$p_i = \frac{A}{(C+i)^\theta}$$

La modificación de Mandelbrot fija de forma más adecuada la distribución de los valores más frecuentes del vocabulario ($i < 100$) [Mon01]. La ley generalizada puede reescribirse como:

$$p_i = \frac{A}{(1+C_i)^\theta} \tag{3.1}$$

donde el parámetro C necesita ser ajustado y no puede tender a 0. En este caso, el factor de normalización A, dependiente de los parámetros C y θ, se define como:

$$A = \frac{1}{\sum_{i\geq 1} \frac{1}{(1+C_i)^\theta}} = \frac{1}{\zeta_C(\theta)} \tag{3.2}$$

3.1.1. Elección del alfabeto fuente

Los compresores de texto desarrollan sus modelos de representación considerando las propiedades del lenguaje natural caracterizadas por las leyes anteriores. Esto supone fijar un *alfabeto fuente orientado a palabras* [Mof89, HC92], aceptando que éstas plantean la unidad mínima de información desde la perspectiva humana. Este tipo de modelos obtienen una representación más realista de las propiedades del texto, considerando que un alfabeto fuente de palabras refleja mejor la verdadera entropía del texto que un alfabeto de caracteres [BYRN99]. A su vez, sobre este tipo de modelos se facilita la detección de las correlaciones de alto nivel existentes entre las palabras [HC92].

Sin embargo, se plantea una nueva cuestión respecto a la elección del alfabeto fuente: la necesidad de representar tanto las palabras como los separadores que componen el texto. A continuación se plantean las tres alternativas generales para el modelado de los separadores en el contexto de la compresión textual:

Alfabeto único. La solución más sencilla, de las tres consideradas, se basa en la representación de palabras y separadores como símbolos de un mismo alfabeto fuente. Sin embargo, esta idea no maneja una propiedad importante del lenguaje natural: la *alternancia* entre palabras y separadores. Este hecho supone la obtención de una función de distribución general para la probabilidad de ocurrencia de palabras y separadores que oculta las propiedades específicas de cada una de las categorías.

A su vez, la elección de un alfabeto único supone representar palabras y separadores en un mismo modelo lo cual dificulta identificar la posible correlación existente entre palabras consecutivas en el texto.

Alfabetos separados. La utilización de alfabetos separados surge como respuesta a las debilidades identificadas en el caso anterior. Esta propuesta considera sendos alfabetos disjuntos para palabras y separadores. Esto facilita la representación implícita de la alternancia existente entre palabras y separadores: una vez se conoce que la codificación del texto comienza con la utilización de uno de los alfabetos, el resto de pasos se lleva a cabo sin más que elegir el alfabeto complementario al considerado en el paso anterior.

Aunque existen algunas propuestas significativas cuya construcción se plantea sobre alfabetos separados [BSTW86, Mof89], este tipo de modelos no consideran una propiedad importante del lenguaje natural: en una gran mayoría de los casos, la palabra está seguida de un *único espacio en blanco* que puede ser asumido como separador por defecto.

Spaceless words. Moura, *et al.* [dMNZBY00] estudian la propiedad anterior sobre diferentes textos de la colección TREC-3[1], obteniendo que el 70 %-80 % de los separadores, utilizados en estos textos, son espacios en blanco únicos. Esta propiedad justifica el hecho de aceptar dicho símbolo como separador por defecto. De esta manera, si una palabra está seguida por un único espacio en blanco sólo la palabra será codificada. En el caso contrario, se codifican tanto la palabra como el separador que la sigue. Por su parte, el descompresor decodifica el texto, palabra a palabra asumiendo, la existencia de un espacio en blanco separador salvo en aquellos casos en los que la palabra decodificada sea un separador.

Esta propuesta de representación de los separadores, denominada *spaceless words*, plantea una transformación del texto original sobre un alfabeto único del que se excluye el espacio en blanco. Esto implica que el modelo actual tampoco gestiona directamente la alternancia palabra-separador. Sin embargo, la eliminación del espacio en blanco permite obtener una mejor representación de la semántica del texto facilitando la identificación de las correlaciones existentes entre las palabras. Además, al excluir el espacio en blanco del alfabeto fuente, se reduce entre un 35 % y el 40 % el número total de símbolos representados con la consiguiente mejora que esto supone en la efectividad de la técnica de compresión. En términos prácticos, el modelado basado en *spaceless words* se acepta como estándar de facto en el modelado de lenguaje natural [Cul08].

La figura 3.1 muestra un ejemplo de representación de un texto (`for a rose, a rose is a rose`) sobre los tres ejemplos de modelado anteriores. Considerando las definiciones dadas de palabra y separador, el presente texto está formado por 15 símbolos. En todos los casos se obtiene una codificación basada en el clásico código de Huffman (§3.3.1).

La figura (1) muestra el árbol de Huffman obtenido sobre un modelo de alfabeto único. En este caso puede observarse como el alfabeto fuente está formado por seis símbolos diferentes: $\Sigma = \{$“`for`”, “`a`”, “`rose`”, “`is`”, “`␣`”, “`,␣`”$\}$, donde “`␣`” se utiliza como representación del espacio en blanco. En la parte inferior de la figura puede verse la secuencia codificada, de acuerdo al presente código, cuya longitud total es de 35 bits (2, 33 bits/símbolo).

La figura (2) muestra el resultado obtenido sobra una representación de alfabetos separados. En este caso se precisan dos árboles de Huffman complementarios para palabras y separadores, de tal forma que los dos alfabetos considerados se describen como: $\Sigma_p = \{$“`for`”, “`a`”, “`rose`”, “`is`”$\}$ y $\Sigma_s = \{$“`␣`”, “`,␣`”$\}$. La secuencia codificada requiere ahora 22 bits, lo que supone 1, 46 bits/símbolo.

Finalmente, la figura (3) muestra el árbol obtenido al excluir la representación del espacio en blanco. De nuevo se opera sobre un alfabeto único cuyo tamaño se reduce en una unidad respecto al utilizado en (1). La secuencia se codifica, en el caso actual, con un total de 21 bits (1, 4 bits/símbolo).

[1] http://trec.nist.gov/

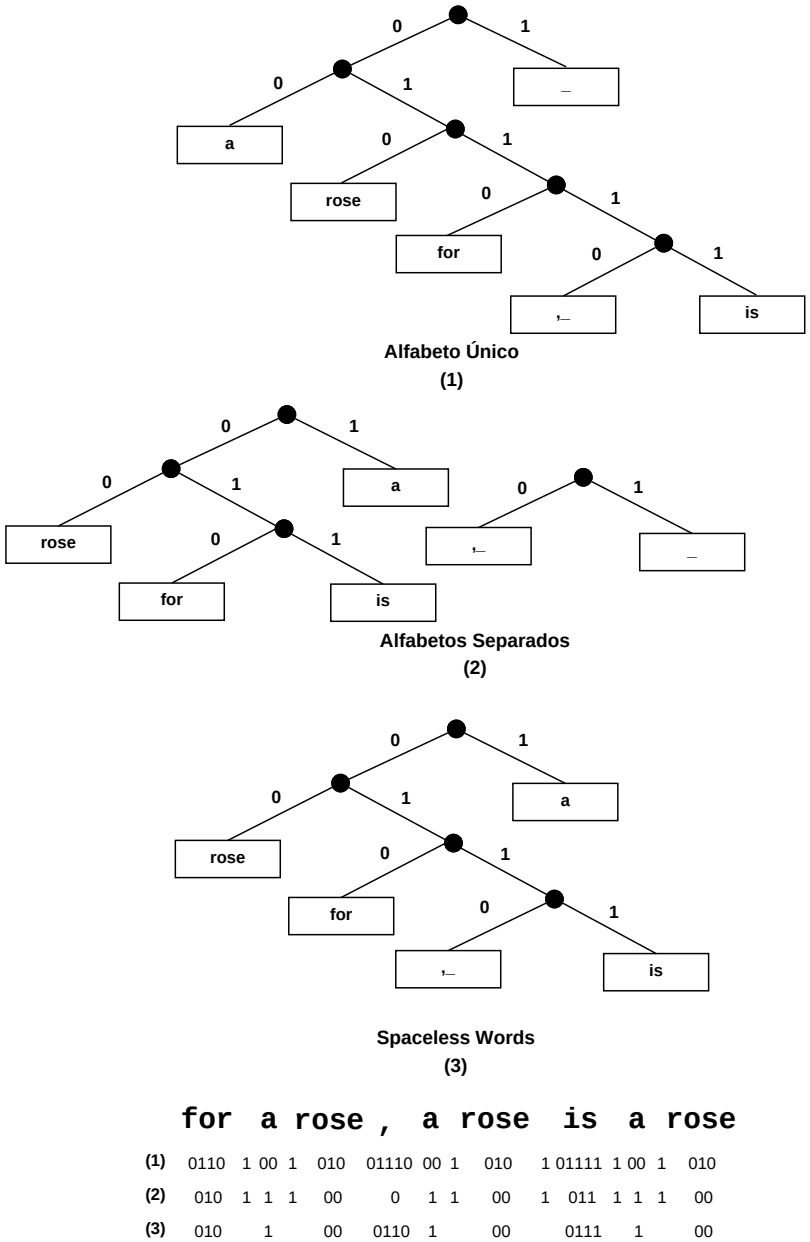


Figura 3.1: Ejemplo de modelado de los separadores sobre un árbol de Huffman.

Finalmente, es importante mencionar que los modelos basados en palabras necesitan textos de gran tamaño para mostrar su verdadera efectividad. De acuerdo a Ley de Heaps, el impacto del tamaño del vocabulario en el resultado final tiende a degradar las ratios de compresión obtenidas al comprimir textos de pequeño tamaño sobre un modelo orientado a palabras.

3.2 Modelado

Esta sección plantea una visión general de las técnicas de modelado de carácter adaptativo y semi-estático. Descartamos profundizar en los modelos estáticos dado que no plantean una solución efectiva para la compresión de lenguaje natural dada la dificultad de prefijar un modelo capaz de representar las propiedades de este tipo de textos.

3.2.1. Modelado adaptativo

Los sistemas de compresión basados en modelado *adaptativo* o *dinámico* procesan el mensaje de entrada en una única pasada en la que el modelo se construye de forma progresiva y el texto se codifica de acuerdo al estado de dicho modelo en cada paso del proceso. Esto supone la actualización incremental del conocimiento relativo a la distribución del texto, facilitando la adaptación del modelo (y, por consiguiente, de la codificación) a las propiedades particulares de dicho texto.

Los modelos adaptativos, en comparación con los semi-estáticos, mantienen una representación más efectiva del texto dado que únicamente reflejan las propiedades derivadas de la parte del texto procesada hasta ese momento. Por su parte, los modelos semi-estáticos se construyen sobre las propiedades globales del texto lo cual puede ser una opción menos efectiva para diferentes partes del mismo, caracterizadas por distribuciones de probabilidad locales diferentes a las obtenidas en términos globales. Considerando que la codificación se adapta al estado del modelo en cada paso del proceso, la representación de un mismo símbolo fuente se puede llevar a cabo con palabras de código diferentes. Este hecho permite ajustar la representación de cada símbolo a la significatividad relativa que éste presente respecto a su contexto de codificación, lo que tiende a reducir, en general, la longitud media de palabra de código utilizada y, por consiguiente, mejorar la efectividad de una técnica adaptativa respecto a una semi-estática. Además, en favor de la efectividad de las técnicas adaptativas, debe considerarse que estas no precisan suministrar explícitamente el modelo utilizado en compresión dado que el descompresor puede reconstruirlo fácilmente sin más que partir de las mismas consideraciones planteadas en el compresor. Sin embargo, como se indica en la sección anterior, no hay una forma absoluta de establecer que una opción es mejor que la otra.

Aunque las técnicas adaptativas tienden a ser utilizadas en numerosos contextos, su foco de aplicación más significativo se define en torno a las procesos de transmisión en tiempo real. La tesis de Antonio Fariña [Far05] plantea una breve descripción de estos procesos a partir de su configuración en cuatro etapas bien diferenciadas: *compresión*, *transmisión*, *recepción* y *descompresión*. Los dos primeras las desempeña el emisor mientras que los dos últimas son llevadas a cabo por el receptor. Un ejemplo sencillo lo representa la descarga de un fichero comprimido desde un servidor web en el que previamente la información se procesa y comprime para su transmisión. En el otro lado de la red de comunicación, una vez la transmisión ha finalizado, el receptor descomprime dicha información para su utilización posterior. Un paso más allá del presente ejemplo se pueden encontrar los escenarios de transmisión en tiempo real en el que las cuatro etapas anteriores se llevan a cabo en concurrencia, del tal forma que los procesos de compresión y transmisión, en el origen, se ejecutan de forma simultánea a la recepción y descompresión en destino. En estos casos, las técnicas de compresión adaptativas plantean la opción más competitiva dado que no necesitan operar de forma previa sobre el mensaje y, por tanto, pueden modelarlo y codificarlo como un flujo continuo de datos.

Las técnicas adaptativas más utilizadas corresponden con la familia de compresores `LZ` (§3.5.1) y otros, como `PPM` (§3.4.1), relacionados con codificación aritmética (§3.3.2).

Los compresores basados en modelos adaptativos deben enfrentar dos problemas fundamentales que inciden directamente tanto en la eficiencia como en la efectividad de la técnica construida. Por un lado, la *actualización del modelo* plantea la necesidad de gestionar, de forma dinámica, la caracterización estadística de los símbolos dentro de sus respectivos contextos de representación y codificación. Este aspecto afecta de forma directa a la eficiencia de la técnica dado que el constante proceso de actualización puede degradar la velocidad de cómputo en compresión y descompresión. Por otro lado, los compresores adaptativos tienen que afrontar la decisión de cómo *incluir nuevos símbolos en el modelo*. De acuerdo a lo planteado en [Adi05], existen dos formas de tratar este problema denominadas, respectivamente, sistemas de compre-

sión con un *modelo plano* y con un *modelo vacío*. La primera de ellas considera, a priori y sobre una perspectiva de equiprobabilidad, todos los símbolos que forman el alfabeto fuente. Esta alternativa se transforma en una solución impracticable en la compresión de texto orientada a palabras donde se debería diseñar un modelo con todas las palabras que podrían ser utilizadas en el texto. Por su parte, los sistemas de modelo vacío plantean una forma dinámica de añadir nuevos símbolos al modelo que requiere establecer un método de asignación de probabilidades para la representación de símbolos noveles en el texto. Esta situación se trata directamente a través del *problema de la frecuencia cero*.

El problema de la frecuencia cero. Autores como Cleary y Teahan [CT95] consideran que el *problema de la frecuencia cero* [Rob82, CW84b, Mof90, WB91, HV92, MSWB94, CT95] tiene un origen filosófico en la "*Crítica de la razón pura*" planteada por Kant. Desde la perspectiva de un modelado adaptativo, este problema se transforma en la decisión de cómo codificar un símbolo no visto anteriormente en el mensaje de entrada. A pesar de que su frecuencia de aparición es 0, se debe asignar una probabilidad positiva que permita su codificación. Sin embargo, la ausencia de un conocimiento, a priori, de la distribución de este tipo de eventos impide tener una base que permita elegir entre diferentes soluciones.

La regla de sucesión de Laplace, originaria de 1812, plantea una forma de estimar la probabilidad subyacente a diferentes eventos no observados dentro de un conjunto de tamaño finito y conocido. Para el caso actual, este conjunto representa el alfabeto de la fuente de información. La primera debilidad de este método radica en la asunción planteada sobre el tamaño del alfabeto. Por ejemplo, el caso de un alfabeto de palabras utilizado para la construcción de un mensaje textual tiene un tamaño indeterminado. Esto supone que la predicción de la ocurrencia de una nueva palabra en el texto, sobre la regla de Laplace, asignará una probabilidad inadecuada al evento de tal forma que éste será codificado de una manera poco efectiva.

A continuación se plantea dicha regla de Laplace y su generalización. A partir de este conocimiento se detallan algunos de los métodos *ad hoc* utilizados en diferentes técnicas reales de compresión adaptativa.

Supongamos un recipiente que contiene N bolas blancas y negras, de las cuales n han sido ya extraídas y c de ellas eran bolas de color negro. Con esta información se plantea cómo estimar la probabilidad de que la siguiente bola extraída sea de color negro. Para valores grandes de c y n esta cuestión no es problemática dado que estimar la probabilidad de aparición como c/n plantea una buena aproximación a la realidad. Sin embargo, para valores pequeños de c y n, sobre todo para valores $c = 0$ o $c = n$, utilizar el valor c/n no sería una buena estimación de la verdadera probabilidad de que la siguiente bola extraída sea negra. Si, por ejemplo, se han extraído solo dos bolas del recipiente y ambas han sido negras, el estimador c/n daría una probabilidad 1 de forma que podría asumirse que todas las bolas en dicho recipiente son negras. De esta manera, la regla de sucesión de Laplace estima la probabilidad del siguiente evento como:

$$\hat{p} = \frac{c+1}{n+2}$$

La regla anterior puede ser generalizada más allá de un conjunto binario de eventos. Consideremos que el recipiente anterior contiene ahora N bolas de q colores diferentes, de forma que C_1 pertenecen al primer color, C_2 al segundo y así sucesivamente. Esto supone que $N = C_1 + C_2 + \ldots + C_q$.

La igualdad anterior supone que la suma de las probabilidades, asociadas a la bola de cada color, tiene que ser igual a 1. Asumimos una distribución equiprobable para el hecho de que una bola pertenezca a un color. Esto supone que todos los valores C_i, $\forall\ 1 \leq i \leq q$, son iguales. Supongamos que en un determinado muestreo del recipiente se han extraído n bolas diferentes

Método	Prob. símbolo novel	Prob. símbolo s
A	$1/(n+1)$	$c_i/(n+1)$
B	r/n	$(c_i-1)/n$
C	r/n	$((n-r)/n)\cdot(c_i/n)$
D	$r/2n$	$(2c_i-1)/2n$
X	t_1/n	$((n-t_1)/n)\cdot(c_i/n)$
C'	$r/(n+r)$	$c_i/(n+r)$
X'	$(t_1+1)/(n+t_1+1)$	$c_i/(n+t_1+1)$

Tabla 3.1: Métodos de estimación de probabilidades de símbolos nuevos, donde n es el número de símbolos codificados hasta el momento, r es el número de símbolos diferentes encontrados, c_i es el número de ocurrencias del i-ésimo símbolo y t_1 el número de símbolos que han aparecido una sola vez, es decir, $t_1 = |\{s \ / \ p_s = 1\}|$.

de las cuales c_1 corresponden al color 1, c_2 al color 2 y así sucesivamente hasta c_q bolas de color q. La probabilidad de que la siguiente bola sea del i-ésimo color podría ser calculada como:

$$\hat{p_i} = \frac{c_i + 1}{n + q}$$

Esto podría ser aplicado al modelado adaptativo de una fuente de información siempre que se su alfabeto fuente tuviese un tamaño finito y conocido a priori (q). Si en un determinado paso del proceso de compresión se han identificado r símbolos diferentes ($r \leq q$), existen otros $q - r$ símbolos aún no identificados y, por tanto, su contador de ocurrencias c_i tiene valor 0. La probabilidad de que el siguiente símbolo no haya sido previamente identificado se obtiene como $(q - r)/(n + q)$.

El método inicializa todos los contadores c_i a 1 de forma que cada símbolo i es tratado como si hubiese sido encontrado previamente $c_i + 1$ veces para obtener un total de $n + q$. Esta propuesta representa la base del método A propuesto por Cleary y Witten [CW84a] en el que se plantea el cálculo sobre un alfabeto de tamaño desconocido. Esto supone que la probabilidad de encontrar un nuevo símbolo se calcula como $\hat{p} = \frac{1}{n+1}$ mientras que la probabilidad de ocurrencia de un símbolo ya encontrado pasa a ser $\hat{p} = \frac{c_i}{n+1}$. En este mismo trabajo, los autores contrastan teóricamente que el método A tiende a obtener un mejor comportamiento que la regla de Laplace generalizada para una gran mayoría de casos reales. Los mismos autores plantean el método B [CW84b] como respuesta a la baja probabilidad asignada a la detección de un símbolo novel. Este nuevo método mantiene un símbolo en un estado de "no identificado" hasta que es encontrado al menos dos veces. Esto supone una reformulación de las probabilidades anteriores que puede ser encontrada en la tabla 3.1 junto a las asociadas con el resto de métodos analizados en la presente sección. Moffat [Mof90] propone el método C sobre las propiedades de los métodos anteriores. Dicho método considera deseable aumentar la probabilidad del símbolo novel con el incremento del número de símbolos encontrados (y, por tanto, de la propia frecuencia del símbolo novel). El mismo autor plantea, en el método C', una revisión en la probabilidad estimada para el símbolo novel, que en este caso es ligeramente inferior a la deseable. El método D [HV92] permite obtener pequeñas, pero consistentes, mejoras respecto al método C cuando se utiliza en modelos basados en PPM. Finalmente, Witten y Bell [WB91] describen diferentes métodos basados en procesos de Poisson cuyo funcionamientomejora el de todos los anteriores. El método X muestra una debilidad para el caso $t_1 = 0$ ante lo cual se propone el método X'. Moffat y Turpin plantean [MT02] una descripción comparativa sobre los métodos presentados.

El término *símbolo (código) de escape* se utiliza para referir a la palabra de código utilizada

para representar la sucesión del evento de símbolo novel. Esta misma terminología se utiliza, en el presente trabajo, para representar aquellas situaciones en las que una nueva palabra se añade en un determinado contexto de representación.

3.2.2. Modelado semi-estático

Los compresores basados en modelado semi-estático plantean un proceso en dos etapas. En la primera de ellas obtienen una representación estadística del mensaje de entrada que se utiliza para la construcción del modelo usado en la etapa compresión. La segunda etapa ejecuta la codificación del mensaje sin más que sustituir cada símbolo fuente por la palabra de código asignada en el modelo. Esto supone que, a diferencia de los compresores adaptativos previamente explicados, cada símbolo del alfabeto siempre se representa con la misma palabra de código. Por lo tanto los compresores semi-estáticos comparten con los estáticos la propiedad de utilizar siempre la misma palabra de código para la representación de los símbolos del alfabeto dado que el estado del modelo no evoluciona a lo largo del proceso de codificación. Sin embargo, los compresores semi-estáticos construyen el modelo de acuerdo a las propiedades específicas del mensaje mientras que los estáticos prefijan unas características globales para todos los posibles mensajes emitidos por una fuente de información. Este es el motivo por el que un compresor semi-estático precisa transmitir, explícitamente, al descompresor el modelo utilizado con el fin de sincronizar los procesos de compresión y descompresión.

La necesidad de llevar a cabo dos pasadas sobre el mensaje de entrada dificulta la utilización de técnicas semi-estáticas sobre flujos continuos de información. Sin embargo, la utilización de un modelo de codificación fijo otorga a las técnicas semi-estáticas un conjunto de propiedades esenciales que permiten su utilización en entornos relacionados con las bases de datos de texto y la recuperación de información. El resultado obtenido con una técnica semi-estática permite *acceso* y *búsqueda* directa sobre el texto comprimido.

Acceso directo. La propiedad de acceso directo describe la posibilidad de acceder aleatoriamente a cualquier posición del texto comprimido e iniciar el proceso de descompresión a partir de ella.

La descompresión representa la base para la construcción de la operación de visualización de resultados en un entorno de recuperación de información. En numerosos casos no se requiere la descompresión completa de un documento sino, únicamente, la de pequeños fragmentos dentro de él (*snippets*). Por ejemplo, la lista de resultados obtenidos ante una determinada búsqueda puede ser contextualizada descomprimiendo un pequeño fragmento del texto alrededor de cada ocurrencia del patrón solicitado.

Las técnicas estáticas también permiten el acceso directo al texto comprimido, sin embargo las adaptativas no. Nótese como la actualización constante del modelo impide descomprimir un fragmento sin procesar previamente el texto anterior dado que la codificación del fragmento deseado se lleva a cabo de acuerdo al estado del modelo obtenido sobre las estadísticas del texto precedente.

Búsqueda directa. Esta propiedad garantiza la posibilidad de buscar directamente sobre el texto comprimido. Esto supone que la descompresión del texto sólo se requiere para propósitos de visualización de resultados, obteniendo un ahorro importante en el tiempo de CPU necesario para la ejecución de las operaciones de búsqueda.

La localización de todas las ocurrencias de un patrón, sobre un texto comprimido, precisa buscar la representación comprimida de dicho patrón de acuerdo al modelo utilizado para la compresión global del texto. Esto se consigue, de forma sencilla, sin más que sustituir cada

símbolo contenido en el patrón por su palabra de código correspondiente. El conjunto de palabras de código que representan el patrón se suministra a un algoritmo de *pattern-matching* secuencial capaz de detectar las ocurrencias del patrón solicitado. Moura, *et al.* [dMNZBY00] señalan que la búsqueda sobre texto comprimido mejora, con algunas técnicas, el rendimiento obtenido por los procesos de búsqueda equivalentes ejecutados sobre el texto original. Por ejemplo, demuestran que usando una codificación de `Huffman` orientada a byte sobre un alfabeto de palabras, las búsquedas sobre el texto comprimido pueden ser hasta 8 veces más rápidas que sobre el texto en plano.

Al igual que en la propiedad anterior, los compresores estáticos permiten búsqueda directa sobre el texto comprimido mientras que los compresores adaptativos requieren un procesamiento completo del texto que permita la actualización del modelo y, de acuerdo con ello, la del propio patrón de búsqueda. A pesar de las dificultades inherentes al diseño de un esquema de compresión adaptativo que soporte de forma eficiente operaciones de búsqueda sobre el texto comprimido, Brisaboa, *et al.* sugieren variantes dinámicas de los códigos `ETDC` y `SCDC` (§3.3.3) que lo permiten [BFNP05, BFNP08, BFNP10].

La conjunción de las propiedades de acceso y búsqueda directa sobre el texto comprimido posicionan a las técnicas de compresión semi-estáticas como una alternativa competitiva para su uso en entornos de recuperación de información. Estas posibilidades de operación permiten obtener un importante ahorro en espacio de almacenamiento y, a su vez, un aumento en el rendimiento de los procesos de búsqueda.

A continuación se plantea una breve explicación del contexto de *pattern-matching* y un repaso de los algoritmos relevantes en él de acuerdo a las necesidades del presente trabajo.

Pattern-matching sobre texto comprimido. Un proceso de *pattern-matching* contempla la búsqueda de las ocurrencias de un patrón $\mathcal{P} = p_1 p_2 \dots p_m$ sobre un texto $\mathcal{T} = t_1 t_2 \dots t_n$, donde $\mathcal{T}$ y $\mathcal{P}$ son secuencias construidas sobre un alfabeto Σ de tamaño finito σ [BM77, KMP77, Sun90]. De acuerdo a lo planteado en [Far05], los procesos de *pattern-matching* puede ser clasificados en seis categorías diferentes aunque, de acuerdo a lo tratado en la presente tesis, nuestro interés se centra exclusivamente en el *pattern-matching exacto*.

A continuación se repasan dos técnicas conocidas en el campo: por un lado, el algoritmo de `Boyer-Moore` [BM77] plantea el representante principal de una familia utilizada para la búsqueda sobre texto comprimido, mientras que el algoritmo de `Horspool` [Hor80] es la técnica seleccionada como base para la implementación de las operaciones de búsqueda presentadas en la Sección §6.4.4. La explicación de esta segunda técnica se complementa con su variante multipatrón [NR02], denominada `Set-Horspool`, también utilizada en las operaciones citadas.

Algoritmo de Boyer-Moore. El algoritmo de `Boyer-Moore` [BM77] se basa en la utilización de una ventana, del mismo tamaño (m) que el patrón, que se desplaza a lo largo del texto comprimido. Inicialmente, el algoritmo alinea la ventana con la primera posición del texto, cuyo recorrido se lleva a cabo mediante un proceso de comparación entre el contenido de la ventana y el patrón solicitado. La comparación se ejecuta de derecha a izquierda, desde el último elemento del patrón hasta el primero (o hasta encontrar una diferencia entre el patrón y el texto comprendido en la ventana de búsqueda). En cada paso del algoritmo, la ventana se desplaza tantas posiciones como indique su máximo valor de desplazamiento seguro. Este valor se calcula de acuerdo a tres funciones previamente calculadas para el patrón solicitado.

La figura 3.2 (extraída de [Far05]) plantea una descripción del algoritmo de `Boyer-Moore`. Supongamos que un sufijo maximal μ en la ventana de búsqueda es también un sufijo del patrón

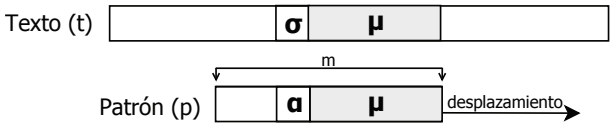


Figura 3.2: Descripción del algoritmo de Boyer-Moore.

solicitado y que, a su vez, el carácter σ en la ventana no coincide con el carácter α en el patrón. Las funciones de desplazamiento se explican como sigue:

- δ_1 representa, para cada sufijo del patrón, la distancia existente desde la posición actual hasta su ocurrencia anterior en p. Esto es, si el sufijo μ aparece en otra posición del patrón (p), δ_1 es la distancia entre α y la ocurrencia anterior de μ no precedida por α. Si, por el contrario, μ no aparece en p, entonces $\delta_1 = m$.

- Si el sufijo μ no aparece en el texto entonces todos los sufijos ν de μ deben ser considerados ya que éstos podrían ser, a su vez, prefijos del patrón en el siguiente paso del algoritmo. En este caso, el valor de δ_2 corresponde con la longitud del prefijo (ν) más largo de p que, a su vez, es sufijo de μ.

- Finalmente, δ_3 está asociado con la última ocurrencia de σ en el patrón (considerando que éste se recorre de derecha a izquierda). Si σ no ocurre en p, entonces $\delta_3 = m$. Esta función permite ahorrar una comparación en el siguiente paso del algoritmo en aquellos casos en los que la ventana sea desplazada sobre δ_1 y σ no haya sido alineada con ninguna otra σ en el patrón.

Dadas δ_1, δ_2 y δ_3, el algoritmo calcula dos valores en cada paso: $M = max(\delta_1, \delta_3)$ y $min(M, m - \delta_2)$, siendo este último valor el utilizado para el desplazamiento de la ventana en cada paso del algoritmo. Nótese como cuando se localiza un resultado en el texto sólo la función δ_2 es utilizada.

Aunque el algoritmo de `Boyer-Moore` tiene un coste $O(m\lambda)$ en el peor caso, su coste promedio es de naturaleza sublineal. Esto supone que su rendimiento mejora en alfabetos de gran tamaño dado que la probabilidad de correspondencias entre los símbolos del patrón y los localizados en la ventana es menor y, por tanto, los desplazamientos tienen una mayor longitud. Esta propiedad es interesante a la hora de aplicar este algoritmo a texto comprimido. Generalmente, el alfabeto utilizado está formado por 256 elementos bajo la consideración de orientación a byte.

Algoritmo de Horspool. El algoritmo de `Horspool` [Hor80] plantea una simplificación del algoritmo de `Boyer-Moore`. Al igual que éste, el presente algoritmo obtiene un mejor rendimiento cuando la longitud del patrón es pequeña respecto al tamaño del alfabeto. La principal diferencia del algoritmo de `Horspool`, respecto a su predecesor, radica en la utilización de una única función de desplazamiento. La figura 3.3 (extraída de [Far05]) plantea una descripción del algoritmo.

En cada paso de procesamiento, este algoritmo compara la última posición de la ventana (β) con la última posición del patrón. Si ambas coinciden, la comparación continúa hacia atrás hasta garantizar la localización de un resultado o, por el contrario, encontrar una diferencia entre el texto en la ventana y el patrón solicitado. La ventana se desplaza de acuerdo a la posición de la siguiente ocurrencia de β.

El algoritmo de `Horspool` comprende sendas etapas de *preprocesamiento* y *búsqueda*. En la primera de ellas calcula la función de desplazamiento para cada símbolo en el alfabeto fuente de acuerdo al patrón solicitado. Si β es la última posición en la ventana de búsqueda, el número de posiciones que la ventana será desplazada es la diferencia entre la última posición en la que

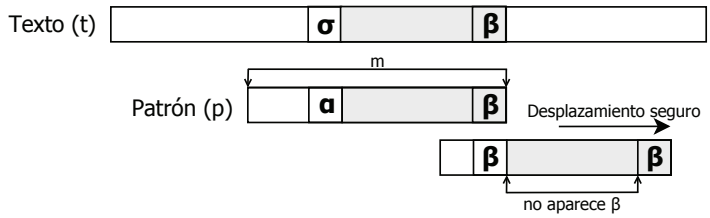


Figura 3.3: Descripción del algoritmo de Horspool.

β aparece en el patrón y el final del propio patrón. Para todos los símbolos del alfabeto que no aparezcan en el patrón la distancia será m. Por su parte, la etapa de búsqueda se ejecuta de forma similar a `Boyer-Moore`, con la diferencia existente en el cálculo del desplazamiento que la ventana lleva a cabo en cada paso del algoritmo.

Algoritmo Set-Horspool. Al igual que `Boyer-Moore`, el algoritmo de `Horspool` puede ser extendido para búsqueda multipatrón (*multi pattern-matching*). Esto supone que el nuevo algoritmo es capaz de localizar, simultáneamente, las ocurrencias de varios patrones de búsqueda diferentes. El presente algoritmo, denominado `Set-Horspool` [NR02], puede considerarse también una simplificación del algoritmo de `Commentz-Walter` [CW79].

Supongamos un conjunto de búsqueda formado r patrones $P = \{p^1, \ldots, p^r\}$. El algoritmo construye un *trie* sobre el orden reverso de los patrones: $P = \{(p^1)^{rev}, \ldots, (p^r)^{rev}\}$ y calcula la función de desplazamiento, de acuerdo a la organización del *trie*, sobre las propiedades del algoritmo original. El procedimiento de búsqueda es similar al original. Esto es, en primer lugar alinea la ventana de búsqueda de acuerdo a la longitud del patrón más corto (l_{min}) y establece como actual el estado inicial del *trie*. La comparación se lleva a cabo de derecha a izquierda sobre la ventana de búsqueda recorriendo, simultáneamente, el *trie*. En el momento en el que una comparación falla, se trata de alinear β con alguna ocurrencia en el *trie*. Si no existe ninguna, la ventana se desplaza l_{min} posiciones. Por otra parte, si en el proceso de comparación se alcanza un estado final del *trie* se habrá localizado la ocurrencia de uno de los patrones solicitados.

El algoritmo `Set-Horspool` es efectivo para búsquedas que contemplen un pequeño número de patrones y se lleven a cabo sobre un texto comprimido haciendo uso de un alfabeto de gran tamaño. Cómo se planteaba anteriormente, las operaciones de pattern-matching consideradas en este trabajo se ejecutan sobre textos comprimidos a partir de un alfabeto orientado a byte.

3.3 Codificación

Aunque el esquema de codificación seleccionado, para un determinado compresor, depende fuertemente de su contexto de aplicación, la presente sección muestra un repaso de aquellos más utilizados en compresión de texto. Hasta aquí se han tratado, exclusivamente, con códigos orientados a bit, pero en la sección actual se presentan también diferentes propuestas de codificación orientada a byte (§3.3.3). Las Secciones §3.3.1 y §3.3.2 repasan los conceptos básicos de codificación de redundancia mínima y aritmética, mientras que la Sección §3.3.4 plantes unas nociones básicas sobre codificación de enteros.

3.3.1. Codificación de redundancia mínima

Los códigos de redundancia mínima (ver definición 2.11) son códigos libres de prefijo cuya longitud media de palabra de código es óptima para un determinado conjunto de probabilidades. Esto

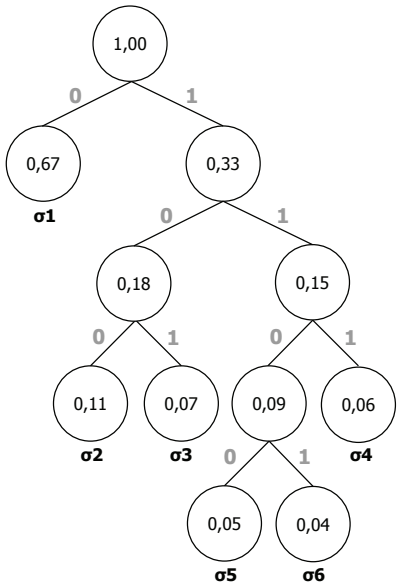


Figura 3.4: Ejemplo de codificación Shannon-Fano.

supone que la longitud de la palabra de código asignada a cada símbolo debe ser inversamente proporcional a su probabilidad en el mensaje.

A continuación se plantean las propuestas de `Shannon-Fano` y `Huffman`. Ambas se basan en la construcción de un árbol de códigos en el que los símbolos más probables aparecen próximos a la raíz mientras que los demás se distribuyen en diferentes niveles de profundidad de tal forma que la distancia entre la raíz y los símbolos crece a medida que decrece la probabilidad de cada uno de los símbolos. La propuesta de `Huffman` construye el árbol desde las hojas a la raíz mientras que la de `Shannon-Fano` lo hace a la inversa, de la raíz a las hojas. Mientras que los códigos de `Huffman` son óptimos para una distribución de probabilidad dada, los códigos de `Shannon-Fano` son ligeramente menos efectivos, por lo que también son conocidos como *códigos subóptimos*.

Codificación de Shannon-Fano. La propuesta de `Shannon-Fano` fue la primera codificación de redundancia mínima planteada a raíz de los trabajos, independientes, de Claude Shannon [Sha48] y Robert Fano [Fan49]. Este algoritmo se basa en la construcción *top-down* de un árbol binario de acuerdo a la siguientes propiedades:

- El conjunto de símbolos debe ser dividido en sendos grupos que acumulen una probabilidad de aparición lo más próxima posible.
- Cada uno de los grupos se sitúa como hijo del nodo raíz, asignando el bit `0` a uno de los grupos y el bit `1` al otro.
- Cada grupo se divide recursivamente hasta que todos los grupos estén compuestos por un único símbolo.

La figura 3.4 muestra un ejemplo de construcción de un código de `Shannon-Fano` para una alfabeto de entrada formado por seis símbolos distribuidos de acuerdo a las siguientes probabilidades: $p(\sigma_1) = 0,67$, $p(\sigma_2) = 0,11$, $p(\sigma_3) = 0,07$, $p(\sigma_4) = 0,06$, $p(\sigma_5) = 0,05$ y $p(\sigma_6) = 0,04$. En el primer paso se obtienen dos grupos con probabilidades $0,67$ y $0,33$. El primero de los grupos está formado, exclusivamente, por σ_1 por lo que se obtiene su palabra de código final: `0`. El segundo de los grupos, identificado con `1`, se divide en sendos grupos con probabilidades $0,18$

y 0, 15. En este caso, como cada uno de los grupos está formado por dos símbolos, son de nuevos subdivididos en el paso final del algoritmo. El código obtenido es: $C(\sigma_1) = \texttt{0}$, $C(\sigma_2) = \texttt{100}$, $C(\sigma_3) = \texttt{101}$, $C(\sigma_4) = \texttt{111}$, $C(\sigma_5) = \texttt{1100}$ y $C(\sigma_6) = \texttt{1101}$.

Codificación de Huffman. David Huffman [Huf52] plantea la primera solución óptima en lo que respecta a códigos de redundancia mínima. Además, caracteriza de una forma más detallada las propiedades que debe tener un código óptimo:

Teorema 3.1 *Dado un alfabeto fuente $\mathcal{A}$, cuyos símbolos siguen una distribución de probabilidad $\mathcal{P}$, existe un código binario de redundancia mínima y libre de prefijo, $\mathcal{C}$, que cumple las siguientes propiedades:*

1. *Si $p_j > p_i$ entonces $|c_j| \leq |c_i|$.*
2. *Las palabras de código que se corresponden con los dos símbolos menos probables tienen la misma longitud.*
3. *Las dos palabras de código más largas son idénticas con excepción del último bit.*

Este teorema permite identificar un código para todos los símbolos del alfabeto fuente, excepto para los dos menos probables. Esto plantea la necesidad de extender el resultado para que incluya estos dos símbolos en el alfabeto. Por lo tanto, se define el alfabeto fuente $\mathcal{A}' = \{x'_1, \ldots, x'_{n-1}\}$ caracterizado por la distribución de probabilidad $\mathcal{P} = \{p_1, \ldots, p_{n-2}, p_{n-1} + p_n\}$. Esto es:

$$p(x'_k) = \begin{cases} p(x_k) & \text{si} \quad k \leq n-2 \\ p(x_{n-1}) + p(x_n) & \text{si} \quad k = n-1 \end{cases}$$

Cualquier prefijo de $\mathcal{A}'$ puede convertirse en un prefijo de $\mathcal{A}$ sin más que añadir un bit a la palabra de código c'_{n-1}. Si el bit añadido es un `0` la palabra obtenida es c_{n-1} mientras que si se añade un `1` se obtiene c_n.

Teorema 3.2 *Si el código libre de prefijo para $\mathcal{A}$ es de redundancia mínima, entonces el código libre de prefijo para $\mathcal{A}'$ también lo es.*

Los dos teoremas anteriores representan la base teórica de la codificación de `Huffman`. Sin embargo, la idea práctica resulta más intuitiva y se centra en la construcción *bottom-up* de un árbol binario a partir de los símbolos que componen el alfabeto fuente:

- Inicialmente se consideran como grupos todos los símbolos individuales que forman el alfabeto fuente siendo su probabilidad la de cada uno de los símbolos.
- En cada uno de los pasos del algoritmo se seleccionan los dos grupos menos probables y se fusionan en uno nuevo caracterizado por la suma de las probabilidades de sus grupos componentes. Sendos bits `0` y `1` son añadidos a los códigos de los símbolos de cada grupo. Cada paso reduce en una unidad el número de grupos.
- El algoritmo se repite hasta que un único grupo acumule la probabilidad de aparición de todos los símbolos del alfabeto. En ese momento, $K(\mathcal{C} = 1)$ (ver ecuación 2.12), lo cual garantiza que el código obtenido es libre de prefijo.

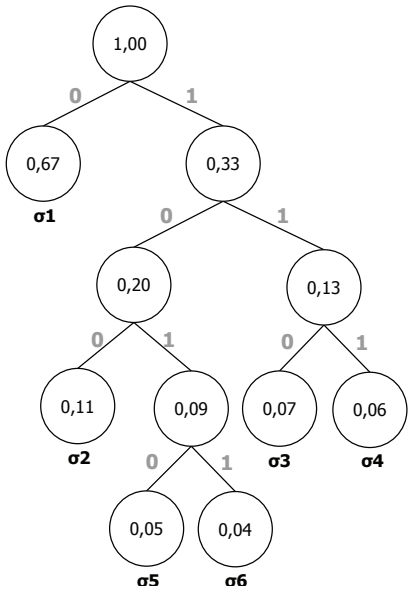


Figura 3.5: Ejemplo de codificación Huffman.

La figura 3.5 muestra el ejemplo de construcción de un código de `Huffman` para la misma distribución de símbolos considerada en la figura anterior. En este caso, la longitud media de palabra coincide con la obtenida por el código de `Shannon-Fano` (1, 75 bits/símbolo). En el primer paso se agrupan los símbolos σ_5 y σ_6 obteniendo un nuevo grupo con probabilidad $0, 09$; el segundo paso une σ_3 y σ_4 dando lugar a un grupo con probabilidad $0, 13$. En el tercer paso puede observarse la unión el grupo de probabilidad $0, 09$ con el símbolo σ_2 obteniendo un grupo de probabilidad $0, 20$. El proceso continúa hasta la fusión final de σ_1 con el grupo que acumula las probabilidades de todos los símbolos restantes en el alfabeto fuente.

En este ejemplo se decide asignar el bit `0` al grupo más frecuente y el bit `1` al menos frecuente. Sin embargo, esta es una decisión arbitraria y la asignación podría llevarse a cabo con cualquier otra política. Esto supone que, para un alfabeto de n símbolos distribuidos de acuerdo a un conjunto de frecuencias determinado, existen 2^{n-1} códigos de `Huffman` diferentes, todos ellos de redundancia mínima.

Schwartz y Kallick [SK64] plantean una clase de códigos de `Huffman`, denominados *códigos canónicos*, cuyas propiedades matemáticas facilitan su almacenamiento y manipulación. Se dice que un código de `Huffman` es canónico si posee la *propiedad de secuencia numérica*:

- Las palabras de código se organizan siguiendo un orden decreciente de tamaño de acuerdo al conjunto de longitudes obtenidas por el algoritmo de `Huffman` y manteniendo un orden lexicográfico creciente para las palabras del mismo tamaño.

- Todos los códigos de un determinado tamaño son números consecutivos.

- Sea c_1 la última palabra de código de longitud l bits (o de mayor valor numérico) y c_2 la primera cuya longitud es $l - 1$ bits (o de menor valor numérico); entonces $c_1 = \mathbb{B}(c_2 - 1)$, donde $\mathbb{B}$ representa la base numérica utilizada para la generación del código de `Huffman`.

La ventajas de este tipo de codificación se estudian de forma detallada en [HL90]. Otros trabajos interesantes al respecto de este tipo de códigos pueden ser encontrados en [MT97, LM07]. Nótese que la implementación del código de `Huffman` orientado a bit planteada por Moffat y Turpin se utiliza como base el desarrollo de algunas de las técnicas planteadas en la presente tesis.

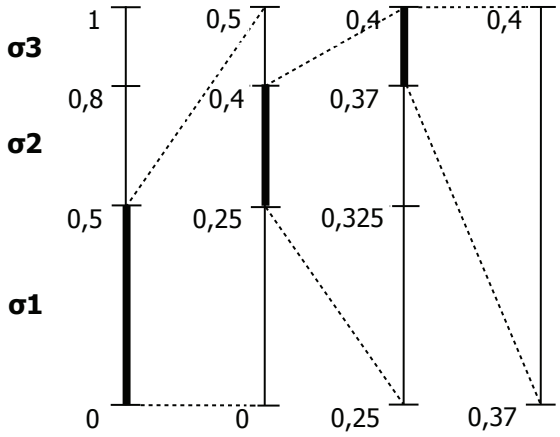


Figura 3.6: Ejemplo de codificación aritmética.

3.3.2. Codificación aritmética

La codificación aritmética, al igual que los códigos de redundancia mínima, está basada en la frecuencia de aparición de los símbolos del mensaje. Sin embargo, su fundamento difiere del anterior dado que la codificación aritmética considera la posibilidad de representar los símbolos con un número fraccional de bits.

La codificación de `Huffman` asigna un número entero de bits para la representación de cada posible símbolo generado por una determinada fuente de información. Sin embargo, de acuerdo a la distribución de probabilidad de dicha fuente, cada uno de estos símbolos podría ser representado con un número fraccional de bits, lo que supone una pérdida de efectividad en la construcción de las palabras de código. Durante mucho tiempo este problema se afrontó a partir de la extensión del alfabeto fuente de tal forma que el código de `Huffman` se aplicaba al nuevo alfabeto extendido. Aún así, la efectividad de esta solución sólo era óptima al considerar una extensión de orden infinito.

Rissanen [Ris76] y Pascoe [Pas76] afrontan la problemática anterior mediante el uso de aritmética de precisión finita como base para la codificación. La figura 3.6 muestra un ejemplo que se utiliza como apoyo a la descripción de la técnica. La codificación aritmética considera el rango de números reales existente en el intervalo $[0, 1)$, de tal forma que asigna a cada símbolo del alfabeto un subintervalo acorde a su probabilidad de aparición. El ejemplo planteado considera un alfabeto de entrada compuesto por tres símbolos $\mathcal{A} = \{\sigma_1, \sigma_2, \sigma_3\}$ con probabilidades $p(\sigma_1) = 0,5$, $p(\sigma_2) = 0,3$ y $p(\sigma_2) = 0,2$. Puede observarse en la figura que el símbolo σ_1 se representa en el intervalo $[0,\ 0,5)$, el símbolo σ_2 en $[0,5,\ 0,8)$ y σ_3 en $[0,8,\ 1)$. El algoritmo de codificación recibe un mensaje de entrada (supongamos $\sigma_1\sigma_2\sigma_3$) y lo codifica mediante un único número real obtenido de acuerdo a las siguientes propiedades.

Cada símbolo en el mensaje se procesa de forma individual reduciendo el intervalo de codificación de acuerdo a su probabilidad de ocurrencia. Dado el primer símbolo, σ_1, se considera el intervalo $[0,\ 0,5)$; σ_2 reduce el intervalo anterior al $[0,25,\ 0,4)$ mientras que con el último símbolo, σ_3, se obtiene el intervalo final de representación: $[0,37,\ 0,4)$. Como puede observarse en este sencillo ejemplo, el intervalo se reduce progresivamente de tal forma que el mensaje se codifica, finalmente, utilizando cualquier número comprendido en el intervalo final. La selección de este valor se lleva a cabo considerando el menor número de bits necesario para su representación. En este caso, el mensaje de entrada podría ser codificado mediante la cadena `011` $(0,375)$ de tal forma que el mensaje inicial se representa con sólo 3 bits.

La efectividad de la codificación aritmética aumenta con distribuciones de probabilidad más

sesgadas. No obstante, esta codificación permite codificar un mensaje $\mathcal{M}$ con una longitud máxima de $\mathcal{H}_0(\mathcal{M})+2$ bits. Esto supone que, mientras un código de redundancia mínima puede perder hasta un bit por símbolo (comparado con la longitud óptima), un código aritmético pierde, como mucho, 2 bits por mensaje.

Witten, *et al.* [WNC87] y Moffat, *et al.* [MNW98] plantean sendas propuestas para la reducción de los costes asociados a la generación de códigos aritméticos mientras que, por otro lado, existen numerosas propuestas relativas a la mejora en los cálculos probabilísticos utilizados para la generación de las distribuciones de probabilidad utilizadas en el contexto actual [Fen94, MNW98, Mof99]. Los códigos aritméticos se utilizan, frecuentemente, en modelos de compresión adaptativos capaces de generar distribuciones de probabilidad muy sesgadas (ver §3.4.1).

3.3.3. Códigos orientados a byte

El conjunto de técnicas de codificación anteriormente repasadas comparten que la construcción de sus palabras de código sigue una orientación a bit. Sin embargo, existe la posibilidad de utilizar otras formas de representación de las palabras de código que, a su vez, mantienen propiedades compresivas sobre el mensaje de entrada. En la sección actual se tratan diferentes técnicas que comparten una orientación a byte para la construcción de las palabras de código.

Los códigos orientados a byte poseen un conjunto de propiedades interesantes para su uso en diferentes contextos de aplicación. Estos códigos son fáciles de obtener y rápidos de decodificar; además, algunos algoritmos pueden obtener representaciones orientadas a byte que favorecen la velocidad de los procesos de *pattern-matching* sobre el mensaje comprimido. Esta mejora en eficiencia se debe, principalmente, a la posibilidad de manejar directamente las palabras de código sin necesidad de realizar operaciones a nivel de bit, más costosas en tiempo de computación [dMNZBY00]. Los códigos orientados a byte se utilizan, tradicionalmente, en aplicaciones relacionadas con la compresión de bases de datos de texto y, por tanto, desarrolladas sobre modelos semi-estáticos en los que el texto se representa como una secuencia de palabras.

Aunque en la presente sección se detallan, únicamente, las propuestas basadas en códigos de `Huffman` y los denominados *códigos densos*, existen diferentes propuestas complementarias en el contexto de los códigos orientados a byte como `Vbyte` [WZ99] o `RPBC` [CM05].

Códigos Plain Huffman y Tagged Huffman. Moura, *et al.* [dMNZBY00] exploran las posibilidades de los códigos orientados a byte, extendiendo el código de `Huffman` original. Esta propuesta, denominada *Plain Huffman Code* (`PHC`), plantea la construcción de un código de `Huffman` sobre un árbol cuyos nodos tienen aridad 256. Esto supone que la longitud de cada palabra de código es siempre un múltiplo de 8 bits. Esta solución representa la mayor aproximación a un código de redundancia mínima orientado a byte. Las ratios de compresión obtenidas por esta técnica, sobre un alfabeto fuente de palabras, se aproximan al 30 %, lo que supone una sobrecarga de, aproximadamente, 5 puntos porcentuales respecto a los resultados obtenidos por Turpin y Moffat [TM97] con un modelo basado en palabras y una codificación de `Huffman` orientada a bit. A cambio, los procesos de descompresión son hasta un 30 % más rápidos gracias a que no requieren operaciones a nivel de bit. Sin embargo, los textos representados con `PHC` sólo permiten acceso directo a través de los bytes que representan el inicio de una palabra de código y, además, están sujetos al problema de los "*falsos positivos*" en procesos de *pattern-matching*.

Los autores plantean un segundo código, denominado *Tagged Huffman Code* (`THC`), cómo respuesta a las debilidades encontradas en el anterior. Este código es similar a `PHC` con la diferencia de que cada byte reserva el bit más significativo para propósitos de marcado. Este bit tiene valor `1` para el primer byte de cada palabra de código mientras que para los restantes su

Valor	ETDC	THC
1	100	100
2	101	101
3	110	110
4	111	111 000
5	000 100	111 001
6	000 101	111 010
7	000 110	111 011 000
8	000 111	111 011 001
9	001 100	111 011 010
10	001 101	111 011 011

Tabla 3.2: Comparación entre THC y ETDC.

valor es 0. Los 7 bits restantes se obtienen de forma similar a cómo se hace en PHC, dado que el bit de marcado no garantiza, por sí mismo, obtener un código libre de prefijo. La efectividad de THC se incrementa hasta ratios de un 35 %, a costa de garantizar propiedades de acceso y búsqueda directa (ver §3.2.2). El bit de marcado permite iniciar la descompresión del texto a partir de cualquier posición (con independencia de que ésta represente un byte inicial o intermedio dentro de una determinada palabra de código). Además, THC facilita la utilización de algoritmos de *pattern-matching* como los planteados anteriormente, dado que el bit de marcado elimina la existencia de falsos positivos. Los experimentos realizados con THC son los que soportan la propiedad anteriormente citada acerca de que las búsquedas sobre texto comprimido son entre 2 y 8 veces más rápidas que sus equivalentes sobre texto plano [dMNZBY00].

Códigos Densos. La familia de códigos densos [Far05] se obtiene sobre una sencilla modificación del código THC. Al igual que éste, los códigos densos reservan el primer bit de cada byte a efectos de marcado y los 7 bits restantes para codificación. Sin embargo, THC marca de forma explícita el primer byte de cada palabra de código, mientras que los códigos densos marcan el último. Esto supone utilizar un 1 como bit de marcado del último byte de la palabra de código mientras que los restantes se fijan a 0. La tabla 3.2 (extraída de [BFNP07]) muestra una comparación de los códigos obtenidos por THC y ETDC para los diez primeros enteros (suponiendo palabras de código de 3 bits).

A pesar de la sutileza del cambio, la repercusión que éste tiene es realmente importante dado que la utilización del bit de marcado garantiza obtener un código libre de prefijo con independencia de los 7 bits restantes. Esto se puede constatar fácilmente con dos palabras de código X e Y tales que $|X| < |Y|$. X no puede ser prefijo de Y dado que su último byte comienza con el bit 1 mientras que $|X|$-ésimo byte de Y tiene su bit fijado a 0. Esta propiedad supone una mejora en la efectividad de los códigos densos dado que éstos pueden utilizar cualquier combinación posible de 7 bits mientras que THC sólo utiliza aquellas generadas a partir del código de Huffman obtenido.

End-Tagged Dense Code (ETDC) [BINP03, Far05, BFNP07] define una técnica semi-estática basada en las propiedades de codificación anteriores. Esto supone disponer de 128 palabras de código de longitud 1 byte, 128^2 palabras de 2 bytes y así sucesivamente. Nótese como la asignación de códigos depende del orden de los símbolos y no de su frecuencia real lo que requiere una ordenación previa del alfabeto de acuerdo a su probabilidad de aparición. De esta manera, los símbolos más frecuentes se codifican con palabras de código más cortas a cambio de añadir al resultado de la codificación, una cabecera que represente el mapeo llevado a cabo sobre el alfabeto original. Los codificación ETDC se ejecuta "al vuelo" sin necesidad de disponer

ningún tipo de almacenamiento explícito (equivalente al árbol de `Huffman` de los códigos) . Esto supone que la codificación de un determinado símbolo se ejecuta de forma sencilla sin más que suministrar, a la función `encode`, el valor entero que lo representa. La decodificación se ejecuta de forma simétrica, mediante la función `decode` responsable de procesar la palabra de código y transformarla en el valor entero que representa.

`ETDC` representa un caso particular de *(s,c) Dense Code*, (referido como `SCDC`) [BFNE03, Far05, BFNP07]). Veamos `ETDC` como una técnica que utiliza 2^{b-1} valores para representar los bytes no finales de una palabra de código y otros 2^{b-1} para los bytes finales. Estos grupos se denominan, respectivamente, *continuers* y *stoppers*, de tal forma que `SCDC` afronta el problema de encontrar el número de *stoppers* (`s`) y *continuers* (`c`) que optimiza la compresión obtenida para un mensaje determinado: $s+c=2^b$. `ETDC` puede ser replanteado, en término de `SCDC`, como $(2^{b-1}, 2^{b-1})$-Dense Code.

La experimentación llevada a cabo con estos códigos fija $s+c=256$ con el fin de utilizar una codificación orientada a byte. La ventaja de `SCDC` respecto a `ETDC` es su habilidad para seleccionar unos valores de s y c óptimos para el texto a comprimir mientras que `ETDC` considera, para todo mensaje, 2^7 *stoppers* y 2^7 *continuers*. La codificación `SCDC` lleva a cabo un paso previo en el cual determina los valores óptimos de s y c y, posteriormente, comprime el texto de acuerdo al código obtenido sobre los valores anteriores.

`SCDC` obtiene unas ratios de compresión ligeramente mejores que las conseguidas por `ETDC`. Ambas técnicas mejoran la efectividad de `THC` y se muestran muy próximas a la obtenida por `PHC`. En lo que respecta a su eficiencia, los códigos densos superan a los basados en `Huffman`, siendo `ETDC` la opción más rápida en compresión y descompresión. En un trabajo de muy reciente publicación [BFL$^+$10], el esquema orientado a palabras utilizado por `ETDC` se extiende con el objetivo de manejar frases como símbolos del vocabulario. Dicha técnica se estudia, con mayor detalle, en la Sección §5.1.

Ambos códigos, `ETDC` y `SCDC`, han sido adaptados para su utilización con modelos dinámicos. `DETDC` y `DSCDC` representan las variantes adaptativas de los códigos anteriores [BFNP05, Far05, BFNP08] que, a su vez, posibilitan la ejecución de procesos de *pattern-matching* sobre los textos comprimidos. Una revisión completa de todos estos códigos se puede encontrar en la tesis de Antonio Fariña [Far05]. Finalmente, se plantean sendas versiones ligeras de los códigos dinámicos anteriores [BFNP10], que facilitan su utilización en receptores con una menor capacidad de cómputo como, por ejemplo, los dispositivos móviles.

3.3.4. Códigos de enteros.

Esta sección plantea una revisión final, a modo de miscelánea, de diferentes técnicas de codificación enfocadas en la representación de números enteros positivos. Estos códigos, también denominados como *códigos estáticos de longitud variable*, plantean una buena solución para aquellos casos en los que el tamaño del alfabeto fuente no puede ser estimado a priori [Cul08]. De forma genérica, los códigos de enteros mapean un número en una secuencia de bits de longitud variable que se utiliza para la representación de dicho número. Estos códigos se utilizan bajo la premisa de que la distribución de probabilidad de los números no es creciente. En caso contrario, se precisa de una reorganización del alfabeto de entrada de tal forma que los números más frecuentes pasen a ocupar los primeros lugares del nuevo alfabeto reordenado y, con ello, ser codificados con palabras de código más cortas.

La *codificación diferencial* (o codificación delta) plantea una forma de preprocesar una lista ordenada de enteros. Se ha utilizado, habitualmente, en la representación de índices invertidos [WMB99] dado que supone una mejora significativa en su compresión al reducir "virtualmente" el tamaño del alfabeto fuente. Esta técnica consiste en representar cada número, en la

secuencia ordenada, mediante su diferencia con el número anterior. Supongamos una secuencia $\mathcal{S} = \{1, 7, 24, 28, 35\}$. Su codificación diferencial mantiene el valor del primer elemento (x_1) y transforma cada uno de los restantes (x_i) de acuerdo a la siguiente operación: $x_i - x_{i-1}$. De esta manera, x_2 sería representado como $7 - 1 = 6$, x_3 como $24 - 7 = 17$ y así sucesivamente. La codificación diferencial de la secuencia anterior es $\mathcal{S}' = \{1, 6, 17, 4, 7\}$, de tal forma que el alfabeto inicial, comprendido en el rango $[1, 35]$, se transforma en un alfabeto distribuido en $[1, 17]$. Esta reducción "virtual" del alfabeto permite llevar a cabo una codificación más efectiva de la secuencia de enteros con cualquiera de las técnicas tratadas a continuación. Nótese como después de aplicar una codificación diferencial, la decodificación del i-ésimo elemento en la secuencia precisa decodificar los $i - 1$ valores anteriores.

La técnica más sencilla para la codificación de enteros es la denominada como *codificación unaria*. Esta representación transforma cada entero x en una secuencia de $x-1$ unos (`1`) seguidos de un cero `0` final. Esto supone, por ejemplo, que el entero 4 se representa con la secuencia de bits `1110`. Esta técnica sólo resulta aceptable para la representación de enteros de pequeño valor.

A continuación se revisan algunas de las técnicas que se han considerado más relevantes para la codificación de enteros. Para profundizar en este tema se recomienda la lectura de [Sal07] donde se plantea un estudio profundo y completo de las diferentes técnicas existentes para la representación de enteros sobre códigos de longitud variable.

Peter Elias [Eli75] plantea una familia de códigos basados en la combinación de dos bloques de bits de longitud variable referidos como *selector* y *localizador*. El selector indica el rango de un conjunto de valores: $1, 2, 4, \ldots, 2^k$ y el localizador representa la posición del valor en el conjunto. El código γ representa el selector en unario y el localizador en binario utilizando un total de $1 + 2\lfloor \log x \rfloor$ bits. Por su parte, el código δ representa el selector mediante su código γ, lo que supone un total de $1+2\lfloor \log \log 2x \rfloor + \lfloor \log x \rfloor$ bits para la codificación de un entero x. Finalmente, el código ω trata de reducir la longitud de los códigos necesarios para representar números mayores. Para ello plantea una codificación recursiva del selector basada en una representación binaria mínima a la que se añade un bit `0` para propósitos de marcado.

El código de Golomb [Gol66] utiliza conjuntos de valores de longitud fija (b) lo que permite mitigar el impacto que supone el crecimiento exponencial del tamaño de los grupos considerados en los códigos de Elías. La codificación de un entero x requiere el cálculo de sendos valores cociente $q = 1 + \frac{x-1}{b}$ y resto $r = x - qb$. Estos valores se codifican posteriormente de forma similar a un código γ, por lo tanto en unario para q y con una codificación binaria mínima para el resto r; el código se obtiene a partir de la concatenación de los códigos anteriores. La elección de b como una potencia de 2 representa un caso especial de los códigos de Golomb, a los que se denomina códigos de Rice [Ric79].

3.4 Métodos de compresión estadísticos

Esta clase de métodos obtienen una representación del texto basada en las propiedades estadísticas de la distribución de los símbolos que lo forman. El cálculo de las estadísticas se lleva a cabo de acuerdo al orden seleccionado para el modelo de representación (ver definición 2.3) que, a su vez, determina la longitud de los contextos de representación considerados. Como se planteaba en la Sección §2.4, el orden del modelo tiene una importancia muy significativa en la caracterización estadística del texto de entrada. La utilización de modelos de orden superior facilita obtener distribuciones de probabilidad más sesgadas al mejorar la calidad de las predicciones que pueden ser realizadas en cada uno de los contextos. Mientras que los modelos de orden 0 calculan la probabilidad de cada símbolo respecto al contexto global representado por el propio texto, los modelos de orden superior obtienen las estadísticas de cada símbolo utilizando como contexto los k símbolos precedentes (donde k representa el orden del modelo).

Los métodos estadísticos ejecutan la etapa de codificación sobre la caracterización estadística obtenida en la etapa de modelado. Esto supone la necesidad de utilizar un código capaz de aprovechar la información aportada en la distribución de probabilidad representada en el modelo. De entre los códigos anteriores, las opciones más efectivas las representan los códigos de `Huffman` y la codificación aritmética.

El propio algoritmo de `Huffman` [Huf52] representa la base de un compresor de naturaleza estadística. Dicho algoritmo precisa una caracterización del alfabeto fuente, $\mathcal{A}$, y de la distribución de probabilidad de los símbolos en el mensaje, $\mathcal{P}$, a partir de los cuales construye el árbol de codificación. Este árbol organiza los símbolos en $\mathcal{A}$ de acuerdo a su probabilidad de aparición en $\mathcal{P}$ por lo que puede ser considerado como un modelo del texto que, posteriormente, se representa siguiendo la asignación de códigos obtenida en el árbol. Estas propiedades muestran el algoritmo de `Huffman` como una técnica de orden 0 dado que la probabilidad de cada símbolo se calcula con independencia de los símbolos que lo preceden. Para lenguaje natural, el algoritmo de `Huffman` obtiene unas ratios de compresión del 65 % al utilizar un alfabeto de entrada orientado a caracteres. Este resultado se debe a la elección del alfabeto de entrada dado que al utilizar una representación basada en palabras la efectividad mejora hasta el 25 % [TM97]. Esta gran mejora demuestra la necesidad de utilizar modelos orientados a palabras para la compresión textual dado que éstos son capaces de identificar la correlación de alto nivel existente entre ellas.

3.4.1. PPM (Prediction by Partial Matching)

El algoritmo `PPM` (*Prediction by Partial Matching*) [CW84b] plantea una propuesta adaptativa de orden superior que obtiene una representación del texto a partir de la combinación de las predicciones realizadas sobre contextos cuya longitud máxima viene determinada por el propio orden del modelo: k. Este tipo de técnicas plantean un importante compromiso entre la efectividad conseguida y el coste de eficiencia que esto supone. El modelado de orden superior que sugiere `PPM` permite obtener unas excelentes ratios de compresión a costa de un aumento significativo tanto en el tiempo de cómputo como en la cantidad de memoria requerida para la construcción y representación de su modelo.

`PPM` utiliza $k+1$ modelos, de órdenes de 0 a k, para la realización de las predicciones con las que codifica cada símbolo en el texto. Dado un símbolo $x \in \mathcal{A}$, `PPM` intenta su codificación utilizando como contexto los k símbolos previos (por lo tanto, sobre el modelo de orden k). Si la transición desde dicho contexto no ha sido previamente encontrada, se emite un código de escape que indica que la codificación pasa a intentarse en el nivel $k-1$. Esto supone considerar los $k-1$ símbolos previos para la codificación de x. El proceso continúa hasta encontrar un modelo en el que la transición pueda ser representada o hasta alcanzar el modelo de orden inferior. Este último caso significa que el símbolo no ha sido previamente identificado en el texto. Por lo tanto, cada uno de los modelos representa los diferentes conjuntos de s símbolos ($\forall s \ / \ 0 \leq s \leq k$) encontrados en el texto y, para cada uno de ellos, almacena las transiciones que representan todos aquellos símbolos a los que preceden en el texto. Cada transición almacena su probabilidad de ocurrencia acumulada en el contexto. Este valor se utiliza como representación del símbolo en su contexto y se codifica aritméticamente. La distribución de probabilidad de cada contexto se actualiza cada vez que dicho contexto se utiliza para la representación de un nuevo símbolo.

El mecanismo de combinación (*blending*) de los modelos de diferente orden conlleva la obtención de distintas variantes del algoritmo `PPM`. Esto se debe a las diferentes políticas que se pueden utilizar en la gestión de los códigos de escape emitidos para afrontar el problema de la frecuencia cero (ver §3.2.1) en `PPM`. En este caso, los símbolos de escape indican los cambios de modelo llevados a cabo desde el modelo de orden superior hasta aquel en que la codificación del símbolo se hace efectiva. En el trabajo original [CW84b], Cleary y Witten consideran la

utilización de los métodos `A` y `B`, mientras que Moffat [Mof90] propone el uso del método `C` y, finalmente, Howard y Vitter [HV94] el método `D`. La aplicación de cada uno de los métodos se lleva a cabo sobre la conceptualización original, siendo el método `D` el que mejor rendimiento plantea [TC97b] para un caso general.

El coste principal de utilizar `PPM` está sujeto a la cantidad de memoria necesaria para la representación del propio modelo. En términos prácticos, el orden de un modelo `PPM` no supera $k = 10$, aunque existen diferentes estudios acerca de la utilización de contextos de longitud ilimitada. Teahan y Cleary [TC97a] proponen el método `PPM`* que explota, completamente, la información contenida en el mensaje a través de todos sus posibles contextos de representación. Los autores reportan una mejora en la efectividad obtenida por `PPM`* respecto a `PPMC` [Mof90], a costa de consumir una mayor cantidad de recursos lo que, a su vez, reduce la velocidad de los procesos de compresión y descompresión.

`PPM` obtiene una efectividad del 20 %-25 % en la compresión de texto. Nótese que estos resultados se obtienen sobre un modelo de caracteres, lo cual demuestra la alta efectividad de esta técnica en comparación, por ejemplo, con las ratios del 65 % reportadas por el código de `Huffman` sobre un alfabeto de caracteres. Sin embargo, a pesar del modelado de orden superior que lleva a cabo, la limitación práctica del valor de k dificulta la identificación de algunas correlaciones existentes entre palabras. Suponiendo que la longitud media de una palabra sea de c caracteres, sería necesario un modelo de orden, al menos, $2c$ para detectar la correlación existente entre dos palabras consecutivas. Este problema se soluciona, de forma sencilla, al utilizar un modelo orientado a palabras. Sin embargo, los alfabetos de palabras tienden a estar formados por un amplio número de elementos. Esto supone que la cantidad de recursos necesarios para obtener modelos de orden superior orientados a palabras se dispara para valores de k superiores a 2, convirtiendo la propuesta en una solución impracticable en términos de espacio y tiempo. Además, por la propia naturaleza adaptativa de `PPM`, el modelo requiere un periodo previo en el que adquirir la información necesaria para obtener una representación efectiva del texto. Esto supone un coste aún más importante si se utiliza un modelo orientado a palabras dado que el número de contextos y relaciones crece exponencialmente con el tamaño del alfabeto fuente.

Aún así, existen algunas propuestas de modelado de orden superior orientado a palabras. La mayoría de ellas concluyen que la utilización de órdenes superiores a 3 no mejoran, notablemente, la efectividad obtenida debido a la propia distribución que siguen las palabras en lenguaje natural. La Sección §5.1 repasa las diferentes técnicas de orden superior orientadas a palabras propuestas en el ámbito de interés del presente trabajo de tesis.

3.5 Métodos de compresión basados en diccionario

A diferencia de las técnicas anteriores, los métodos basados en diccionario no consideran explícitamente las estadísticas de los símbolos en el mensaje. Los métodos actuales construyen una estructura de *diccionario* en la que almacenan cada símbolo del alfabeto fuente que cuya representación se lleva a cabo a través de la palabra de código obtenida a partir de su posición en dicho diccionario. Estas mismas palabras de código se utilizan, en la etapa de codificación, para representar la ocurrencia de cada símbolo en el texto. Este proceso se desarrolla de forma sencilla sin más que sustituir cada símbolo en el mensaje por la palabra de código que lo identifica. Por lo tanto, el diccionario puede observarse como una función de mapeo "uno a uno" entre símbolos y palabras de código. La capacidad compresiva de estos métodos reside en que la longitud media de la palabra de código utilizada para la codificación es menor que la longitud media de los símbolos representados.

Las técnicas basadas en diccionario se utilizan de forma generalizada dada su simplicidad y el buen compromiso (*trade-off*) espacio/tiempo que obtienen. La familia `LZ` representa el conjunto

de propuestas más utilizado entre los compresores basados en diccionario (§3.5.1). Sin embargo existen otros tipos. En la presente sección se muestra especial atención a las técnicas basadas en gramáticas (§3.5.2) cuya propiedad principal radica en la utilización de una gramática libre de contexto (inferida del mensaje representado) para la extensión del diccionario.

Los códigos densos, anteriormente explicados, se pueden considerar como técnicas de diccionario bajo la consideración del mapeo palabra-código que implementan. La asignación de palabras de código, tanto en `ETDC` como `SCDC` (y sus respectivas variantes dinámicas), no se lleva a cabo de acuerdo a la frecuencia de aparición de los símbolos sino siguiendo el orden establecido en el alfabeto fuente de acuerdo a la distribución de probabilidad que éste sigue en el texto modelado. Esto se traduce en una relación biunívoca entre palabras de código y símbolos en el alfabeto fuente, lo cual puede entenderse como una función de mapeo similar a la considerada en las técnicas basadas en diccionario. De esta manera, las ratios de compresión obtenidas por `ETDC` y `SCDC`, entre el 30 %-35 %, pueden utilizarse como una referencia de la efectividad alcanzable utilizando técnicas de diccionario para la compresión de texto.

3.5.1. Compresores LZ (Lempel-Ziv)

Las técnicas de compresión `LZ` surgen del trabajo de Jacob Ziv y Abraham Lempel a finales de los años 70. `LZ77` [ZL77] y `LZ78` [ZL78] supusieron un gran paso adelante en el campo de la compresión. Fruto de estos algoritmos se han obtenido numerosos compresores ampliamente utilizados como `gzip`, `pkzip` o `arj` (basados en `LZ77`) o `compress`, desarrollado a partir de las propiedades de `LZ78`. Más recientemente, Igov Pavlov ha planteado una nueva variante del algoritmo `LZ77`, denominada `LZMA`, que ha sido integrada en el software de compresión `p7zip`[2] junto a otras conocidas técnicas como `gzip` o `bzip2` (todas estas herramientas se encuentran detalladas en el apéndice A).

LZ77. El algoritmo `LZ77` [ZL77] representa la raíz de la presente familia de técnicas basadas en diccionario. Su característica principal es el uso de una *ventana deslizante* de tamaño finito que se utiliza como histórico de procesamiento para la codificación de los siguientes símbolos en el mensaje de entrada (que constituyen el *lookahead buffer*). La capacidad compresiva de `LZ77` radica en que las secuencias encontradas en el *lookahead buffer* hayan sido previamente identificadas en el texto y, por tanto, están representadas dentro de la ventana deslizante. En este caso, cada una de estas secuencias se puede codificar sin más que referenciar su última ocurrencia dentro de la ventana. La codificación se lleva a cabo mediante tripletas de la forma $< p, l, s >$, que también se utilizan en el caso de que el primer símbolo en el *lookahead buffer* no esté representado en la ventana deslizante. El valor p indica la posición (dentro de la ventana deslizante) en la que se localiza la secuencia identificada en el *lookahead buffer*. Dicha posición se referencia como la diferencia entre la posición actual y la posición de inicio de la secuencia en la ventana. El valor l indica la longitud de la secuencia de símbolos identificada mientras que s representa el símbolo siguiente, en el *lookahead buffer*, a la secuencia representada. La ventana se desplaza en l posiciones una vez que la codificación ha sido emitida. Este desplazamiento es de una sóla posición si no se ha encontrado ninguna aparición en la ventana. La figura 3.7 muestra la codificación `LZ77` de la cadena "`abacbaab`". Se utiliza una ventana deslizante de cuatro posiciones.

El paso `(1)` plantea la codificación del símbolo "`a`" que, evidentemente, no está representado en la ventana previamente inicializada a un estado vacío. Se emite el código `<0,0,a>` que indica que el símbolo "`a`" no está representado en la ventana. El paso `(2)` es similar, lo que supone la adición de un nuevo símbolo: "`b`". El paso `(3)` encuentra "`a`" en la primera posición del

[2] http://www.7-zip.org/

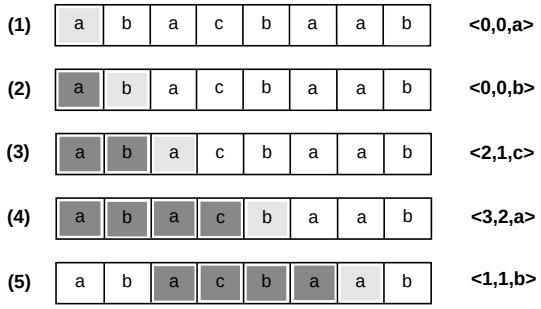


Figura 3.7: Ejemplo de codificación basada en LZ77.

lookahead buffer, lo que permite buscar su ocurrencia anterior en la ventana. Esta se encuentra dos posiciones antes de la actual y tiene una longitud 1. Esto se representa con el triple `<2,1,c>` que indica que la siguiente cadena empieza dos posiciones antes de la actual, tiene longitud 1 y está sucedida por el símbolo "`c`". Nótese como en este paso se han codificado dos símbolos del mensaje, de tal forma que el paso `(4)` afronta la representación del *lookahead buffer* a partir del símbolo "`b`". Recorriendo la secuencia se localiza que la secuencia "`ba`" está en la ventana; esto supone la emisión del triple `<3,2,a>` que indica la representación de la secuencia "`baa`". En el paso `(5)` la ventana ya se ha desplazado lo suficiente como para que los dos primeros símbolos del texto estén fuera de ella. La compresión finaliza con la emisión del triple `<1,1,b>` en representación de la secuencia "`ab`".

La efectividad de un compresor basado en `LZ77` depende del tamaño de ventana considerado. Con tamaños más grandes de ventana se mantiene un histórico mayor de información y, por tanto, se aumenta la probabilidad de que secuencias de mayor longitud (en el *lookahead buffer*) estén representadas en la ventana. Sin embargo, el incremento del tamaño de ventana también supone un aumento en el rango de valores utilizado para la representación de los desplazamientos. Generalmente se utilizan 12 bits para la representación de la posición (lo que supone una ventana de 4096 posiciones) y 4 para la longitud de la secuencia. Además, se debe considerar una longitud mínima de secuencia con el fin de compensar el coste de representación del triple. Generalmente se considera una longitud mínima de 3 caracteres. Esto supone la utilización de 2 bytes para la representación de los pares desplazamiento-longitud a los que hay que sumar un tercer byte para la representación de s.

LZ78. La segunda propuesta de Ziv y Lempel, `LZ78` [ZL78], plantea la utilización de un diccionario general en el que se almacenan todas las frases identificadas en el texto a costa de eliminar la utilización de la ventana deslizante. El nuevo diccionario se implementa sobre un *trie* que permite su recorrido eficiente. El texto de entrada se lee, carácter a carácter, hasta que dado un nodo del trie no existe ninguna arista que permita llevar a cabo la transición al siguiente carácter identificado en el texto. De esta manera, la frase se codifica con un entero que identifica en el diccionario la secuencia de los $k-1$ caracteres localizados en el trie seguido del carácter siguiente para el que aún no existe transición. Por lo tanto, cada frases en `LZ78` se codifica como $< p, s >$. La figura 3.8 muestra la codificación de la cadena anterior "`abacbaab`" de acuerdo a las propiedades de `LZ78`.

El primer y el segundo paso incluyen en el diccionario los símbolos "`a`" y "`b`". El paso `(3)` continúa la lectura del texto identificando el símbolo "`a`" (ya representado en el texto) y, a continuación, el símbolo "`c`". La frase "`ac`" no está representada en el diccionario, por lo que se emite el par `<1,c>` que referencia el símbolo "`a`" en el diccionario y lo concatena con "`c`". El paso `(4)`, identifica el símbolo "`b`" en el *trie* pero, de nuevo, no existe una transición hacia

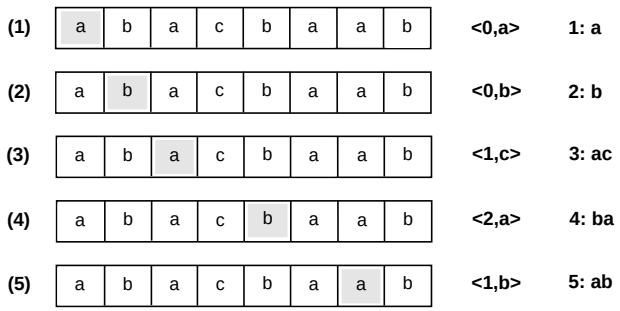


Figura 3.8: Ejemplo de codificación basada en LZ78.

el símbolo "a". La frase "ba" se añade al diccionario y se codifica como <2,a>. Finalmente, se añade la frase "ab" sobre las mismas propiedades referidas en los dos pasos anteriores.

Welch [Wel84] propone una variante de LZ78, denominada LZW, que se utiliza como base para la construcción del compresor compress y en el desarrollo del formato GIF para la compresión de imágenes. La principal diferencia, respecto a su predecesor, es que LZW sólo emite referencias a la posición en la que la secuencia se almacena en el diccionario evitando la emisión de caracteres necesaria en LZ78. Para ello inicializa el diccionario con los símbolos que componen el alfabeto fuente y toma como carácter inicial de la siguiente frase, el último de la anterior.

LZMA. Igor Pavlov integra en el software p7zip una nueva variante del algoritmo LZ77 denominada LZMA (*Lempel-Ziv-Markov chain-Algorithm*). Esta nueva propuesta se presenta como una técnica capaz de alcanzar una alta efectividad sobre procesos de descompresión rápidos y caracterizados por un pequeño consumo de memoria.

LZMA limita a 1GB el tamaño del diccionario, aunque esta capacidad se puede incrementar hasta los 4GB en caso de ser requerido. El presente algoritmo lleva a cabo una codificación basada en la emisión de triples <p,l,s> caracterizados de acuerdo a las siguientes propiedades:

- Si el siguiente símbolo en el *lookahead buffer* no se encuentra en el diccionario se codifica en el intervalo $[0, 255]$.
- En caso contrario se utiliza *range-encoding* [Mar79, Sch98] para la codificación del par longitud-distancia.
- A diferencia de LZ77, el algoritmo actual mantiene un histórico con las cuatro distancias más recientemente utilizadas. Esto permite codificar con sólo 2 bits la distancia actual si ésta se encuentra representada en este conjunto de valores.

El nuevo algoritmo LZMA hace que p7zip sea una técnica más efectiva que sus precursoras, como gzip o pkzip, obteniendo mejoras de hasta el 50 % en colecciones textuales de la TREC3[3]. Sin embargo, los procesos de compresión de p7zip se caracterizan por un importante consumo de memoria y por unos altos tiempos de cómputo. A cambio, mejora a todas sus predecesoras en tiempo de descompresión utilizando un conjunto limitado de recursos determinado por el tamaño del diccionario seleccionado.

3.5.2. Compresores basados en gramáticas

La compresión basadas en gramáticas [KY00, CLL+05] puede considerarse una disciplina particular dentro de las técnicas basadas en diccionario. Estos compresores no construyen un dic-

[3] http://trec.nist.gov/

cionario "plano" con las frases identificadas en el texto sino que generan una gramática libre de contexto específica para la representación del conjunto de frases formadas a partir del mensaje de entrada. La obtención de estas gramáticas se lleva a cabo bajo la consideración de diferentes heurísticas dado que la determinación del conjunto de frases que optimiza la compresión de un texto es un problema $\mathcal{NP}$-difícil [SS82]. Aunque esta sección enfoca de forma detallada la descripción del algoritmo `Re-Pair` [Lar99, LM00] (considerado como base para el desarrollo de la jerarquía de contextos utilizada en la técnica `E-G`$_k$ presentada en la Sección §5.4), en primer lugar se plantea una revisión general del contexto de la compresión basada en gramáticas.

Rubin [Rub76] plantea un sistema de codificación que utiliza, al igual que `Re-Pair`, un procesamiento recursivo basado en la frecuencia de los diferentes bigramas (pares de símbolos) identificados en el texto. Sin embargo, este trabajo se centra en las condiciones de finalización del algoritmo (basadas en la frecuencia o la longitud de la frase) sin optimizar la codificación de la representación obtenida. Por su parte, Apostolico y Lonardi [AL00] proponen el uso de un árbol de sufijos como base para la identificación de frases dentro de la técnica `Off-Line`. Los autores consideran una *función de ganancia* que determina la siguiente mejor frase de acuerdo a una evaluación basada en su longitud, frecuencia y en el coste asociado a la representación de todas sus ocurrencias. El sistema `Ray` [CW02] también plantea un procesamiento *offline*, en múltiples pasadas, sobre el texto. En la primera etapa se obtiene una representación estadística basada en la frecuencia de aparición de los diferentes bigramas identificados en el texto. Las dos siguientes etapas se repiten hasta que ningún bigrama aparezca más de una vez o hasta alcanzar un número máximo de iteraciones. El conjunto de estas etapas afronta tanto la identificación de los símbolos candidatos a ser reemplazados como su propio reemplazo en la siguiente etapa. `Ray` difiere de `Re-Pair` en la forma en la que procesa aquellos bigramas con la misma frecuencia de aparición. Supongamos tres símbolos `abc`; el digrama `ab` sólo se considera para reemplazo si `a` ocurre al menos el mismo número de veces que `c`. En caso contrario, el digrama `bc` es el considerado como candidato en la siguiente etapa del algoritmo. Finalmente, `Sequitur` [NMI97] desarrolla un proceso *online* sobre el que obtiene la gramática de acuerdo a dos propiedades principales: *unicidad del bigrama* y *utilidad de la regla*. La unicidad del bigrama garantiza que ningún par de símbolos pueda ser generado por dos reglas diferentes de la gramática, mientras que la utilidad de la regla prohíbe la formación de reglas que se usen una sóla vez. El resultado de la compresión obtenida por `Sequitur` es una representación de la gramática construida sobre triples similares a los utilizados en `LZ77`.

Re-Pair. El algoritmo `Re-Pair` [Lar99, LM00] considera una representación del diccionario basada en una jerarquía binaria en el que las frases más largas se representan como referencias a sus frases constituyente de nivel inferior. `Re-Pair` guarda ciertos puntos en común con el algoritmo `LZ78` aunque afronta directamente uno de sus puntos débiles: la inclusión en el diccionario de aquellas frases que sólo se utilizan una vez en el texto. Para ello `Re-Pair` construye un modelo completo del texto a partir del cual desarrolla un mecanismo de inferencia sobre el que obtiene la configuración final de la gramática representativa del diccionario. La utilización de este modelo implica la necesidad de mantener, en memoria, una representación completa del mensaje, con el coste espacial que eso supone. A cambio, `Re-Pair` muestra una alta velocidad de descompresión y una efectividad competitiva ante la compresión de tipos de datos tan distantes como el texto o las secuencias biológicas. Además, facilita la búsqueda y acceso directo al resultado comprimido lo cual ha sido satisfactoriamente utilizado en campos como el lenguaje natural [Wan03], las secuencias biológicas [CFMPN10] o la compresión de grafos web [CN07], entre otros.

El mecanismo de inferencia considerado por `Re-Pair` comparte la consideración, planteada por Manber [Man97], de obtener frases a partir de la concatenación de los bigramas de caracteres más frecuentes. Mientras que Manber limita el tamaño máximo del diccionario a 128 frases (bajo

la consideración de que los 128 primeros caracteres son los utilizados por los textos en inglés), `Re-Pair` prosigue con su proceso recursivo de reemplazo de pares de símbolos hasta que ningún bigrama aparece más de una vez en la secuencia resultante. En cada paso del algoritmo se selecciona el bigrama más frecuente cuyas ocurrencias son reemplazadas por un nuevo símbolo que es añadido al alfabeto fuente. Las frecuencias de aparición de todos los bigramas son revaluadas, en cada paso del algoritmo, respecto al nuevo alfabeto extendido con el símbolo generado.

La implementación del mecanismo de inferencia tiene un coste $O(n)$ tanto para la representación del modelo como para el tiempo necesario en la derivación de la gramática final. El mensaje inicial se representa sobre un vector de n posiciones (referido como secuencia) cada una de las cuales requiere tres palabras de memoria: una para la identificación del símbolo representado en la posición y otras dos para referenciar las ocurrencias anterior y posterior del digrama. A su vez, mantiene una cola de prioridad de tamaño $\lceil\sqrt{n+1}\rceil - 1$ que almacena en cada posición una lista de todos aquellos bigramas con k ocurrencias ($2 \leq \sigma \leq \sqrt{n}$). Todos los bigramas con más de $\sqrt{n}$ ocurrencias se almacenan en una misma lista. Cada elemento en las listas contiene un puntero al registro que almacena la información del bigrama desde el que se puede acceder, a través de otro puntero, a su primera ocurrencia en la secuencia. Dado un alfabeto de entrada original $\mathcal{A}$ / $|\mathcal{A}| = \sigma$, el número máximo de bigramas es σ^2 y la ocupación máxima de las listas se produce justo antes del primer reemplazo. En cada paso del algoritmo, la ocupación de las listas se reduce en, al menos, una unidad. Finalmente, una tabla hash almacena tanto los bigrmas constituidos en reglas de la gramática como aquellos que se mantienen como candidatos a lo largo de la ejecución del algoritmo. Una explicación más completa y detallada de la implementación de se encuentra en [Lar99].

El coste de obtener la representación de un texto formado por un total de n símbolos, con un alfabeto de entrada de tamaño $|\mathcal{A}| = \sigma$, a partir del que se infieren k' frases es de $5n+4\sigma^2+4\sigma'+\lceil\sqrt{n+1}\rceil-1$. Por su parte, el descompresor limita, notablemente, las necesidades de espacio dado que únicamente demanda dos palabras de memoria por cada frase en la jerarquía más el coste particular de almacenar el alfabeto de entrada.

El resultado obtenido por `Re-Pair` es una representación comprimida del mensaje de entrada formada por un secuencia de n' símbolos en la que se cumple que ningún bigrama aparece en más de una ocasión. Por otro lado se debe considerar que la gramática obtenida debe ser, explícitamente, suministrada al descompresor considerando que ésta ha sido inferida de forma semi-estática a partir de las propiedades específicas del mensaje procesado.

Raymond Wan [Wan03] estudia las posibilidades de `Re-Pair` para su utilización en entornos de recuperación de información. Su trabajo orienta el algoritmo original hacia una perspectiva basada en palabras, de tal forma que los bigramas considerados pasan a ser pares de palabras. Wan plantea un preprocesamiento del texto que separa éste en sendos flujos de palabras y separadores. El primero de ellos se normaliza a minúsculas y, posteriormente, se transforma mediante un algoritmo de *stemming*. Además, la fase de generación de frases considera los símbolos de puntuación existentes en el texto original de tal forma que las frases añadidas a la gramática representan frases reales en el texto. Esta variante del algoritmo `Re-Pair` original se integra en un sistema de recuperación, denominado `Re-Store`, que facilita la navegación eficiente por las frases del texto comprimido sin renunciar a una alta efectividad.

3.6 Métodos de compresión basados en preprocesamiento

La compresión de datos se ha planteado, tradicionalmente, a través de técnicas estadísticas y basadas en diccionario. Sin embargo, en los últimos años se considera una tercera familia de compresores basada en el preprocesamiento del mensaje de entrada. Aunque dicho preprocesamiento no plantea una forma explícita de compresión, sí es capaz de obtener una transformación

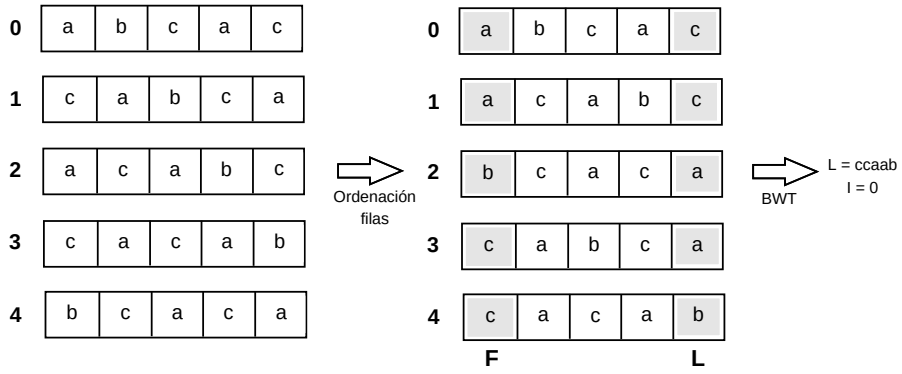


Figura 3.9: Ejemplo de la transformada BWT.

más compresible del mensaje de entrada. La compresión de esta nueva representación permite alcanzar niveles de efectividad competitivos con los obtenidos en técnicas de compresión clásicas.

El método de preprocesamiento más relevante es la transformada de *Burrows-Wheeler* (`BWT`) [BW94]. De forma complementaria y, exclusivamente para compresión de texto, se considera un conjunto de algoritmos de compresión que conjugan un preprocesamiento del texto y su compresión con técnicas universales. A diferencia de la `BWT`, estas transformaciones se basan en detalles específicos del texto que permiten eliminar pequeñas cantidades de redundancia facilitando obtener una pequeña mejora de efectividad respecto a la conseguida por la técnica universal. La Sección §4.1 plantea una profunda revisión de estas técnicas junto con la explicación de un segundo grupo de compresores, también basados en un preprocesamiento del texto, que enfocan la redundancia subyacente a una codificación del texto orientada a byte.

3.6.1. Transformada de Burrows-Wheeler (BWT)

Cómo se planteaba anteriormente, la `BWT` no es una técnica de compresión sino sólo una forma de transformar un determinado mensaje de entrada en una representación equivalente con mejores propiedades para su compresión. La `BWT` es un algoritmo reversible que se ejecuta sobre un mensaje $\mathcal{M} = m_1 m_2 \dots m_n$ de tamaño n y cuyo resultado es un nuevo mensaje $\mathcal{M}'$ con el mismo contenido que $\mathcal{M}$ expresado en un orden diferente. La transformada inversa de *Burrows-Wheeler* es capaz de recuperar $\mathcal{M}$ a partir de $\mathcal{M}'$ y un pequeño conjunto de información extra.

Transformada `BWT` directa. La transformada de Burrows-Wheeler de un mensaje $\mathcal{M}$ construye, en primer lugar, una matriz de tamaño nxn (T_{nxn}) a partir de la rotación circular del mensaje original. Esto supone que $\mathcal{M} = m_1 m_2 \dots m_n$ está en la primera fila, $\mathcal{M}_{>1} = m_n m_1 \dots m_n - 1$ en la segunda y así sucesivamente hasta la n-esima fila en la que se almacena el mensaje $\mathcal{M}_{>n-1} = m_2 m_3 \dots m_1$. A continuación T_{nxn} se ordena lexicográficamente por filas, de tal manera que la fila I almacena el mensaje original $\mathcal{M}$. La figura 3.9 muestra el resultado de los pasos anteriores. Puede verse como las columnas izquierda y derecha se refieren, en la matriz transformada, como `F` y `L`:

- `F` contiene los mismos símbolos que $\mathcal{M}$ pero ordenados alfabéticamente.

- El j-ésimo carácter de `L` precede en $\mathcal{M}$ a la cadena almacenada en la fila j-ésima.

Por lo tanto, el resultado de la transformada `BWT` directa está formado por los símbolos en la columna `L` más el valor I que indica la fila en la que se representa el mensaje original.

Transformada BWT inversa. La transformada BWT inversa utiliza el resultado obtenido en la etapa anterior para la reconstrucción del mensaje original. El primer paso obtiene la columna F sin más que ordenar lexicográficamente los símbolos almacenados en L. Estas dos cadenas, L y F, se utilizan para la construcción de una tercera, denominada T, que almacena la correspondencia existente entre las dos anteriores. Si L[j] almacena la k-ésima ocurrencia del carácter 'b' en L, entonces T[j] = i dado que F[i] es la k-ésima ocurrencia de 'b' en F.

De acuerdo a las propiedades con las que se define T, la relación entre F y L puede ser expresada como F[T[j]]=L[j]. El texto original se restablece utilizando el valor I, L y T sobre un proceso iterativo en n pasos.

Utilización de la BWT para compresión. Cómo ha quedado constatado en la explicación anterior, la BWT no obtiene una compresión del texto de entrada sino una transformación del mismo que permite una compresión más efectiva. Supongamos que la palabra "casa" se repite numerosas veces en $\mathcal{M}$. La transformada directa representaría conjuntamente todas aquellas filas que empiecen por "asa" y que, con una alta probabilidad, terminarán con la letra "c". Esto supone que una región de L mantendrá próximas muchas ocurrencias "c" junto con aquellos otras letras que precedan al sufijo "asa" (por ejemplo "m" o "p", en las palabras "masa" y "pasa"). Esta propiedad puede generalizarse a cualquier cadena encontrada en $\mathcal{M}$, de tal forma que la columna L mantendrá próximos conjuntos de ocurrencias de un mismo símbolo.

Este resultado supone que la probabilidad de un determinado símbolo es muy alta en determinadas regiones de L y muy baja en otras. Esta distribución puede ser aprovechada por un sistema de compresión *move-to-front* (MTF) [BSTW86] que codificaría las ocurrencias de "c" como el número de símbolos diferentes encontrados desde la ocurrencia justo anterior. Esto se traduce en secuencias de ceros como representación de todas aquellas ocurrencias de "c" que aparezcan contiguas en L. Esta nueva secuencia de enteros, obtenida con MTF, puede ser comprimida con un código de Huffman o un aritmético. También puede utilizarse una codificación *Run Length Encoding* (RLE) que obtiene una alta efectividad gracias a las rachas de ceros existentes en la secuencia obtenida con MTF.

La transformada BWT se utiliza como base para la construcción del compresor bzip2 (§A.3.1) orientado a caracteres. Moffat e Isal [MI05] definen una técnica basada en BWT y orientada a palabras para la compresión de lenguaje natural. El texto se representa de acuerdo a un modelo genérico de palabras obtenido de mediante la transformación *spaceless words*. Cada palabra se reemplaza por el valor entero que la identifica y la secuencia obtenida se utiliza como entrada a la BWT. El resultado obtenido es, posteriormente, transformado mediante operaciones *recency-rank* en vez de MTF dado que la distribución de símbolos en L no presenta las mismas propiedades para palabras que para caracteres. El resultado final se codifica con Huffman obteniendo unos ratios de compresión cercanos al 20 %.

Hay ciertas cosas
que deben pasar,
qué sabio es el destino ...
Tontxu Ipiña

4

Construcción de Diccionarios Textuales para Compresión Universal

Existen dos puntos de vista complementarios para llevar a cabo la compresión de texto [SGD05]. Por un lado, los compresores *ad hoc.* para lenguaje natural se construyen directamente sobre las regularidades de este tipo de textos. Por otro lado, existe la posibilidad de utilizar algoritmos de preprocesamiento que obtienen una transformación reversible del texto caracterizada por una mayor redundancia que el texto original. Esta segunda alternativa favorece la compresión final del texto preprocesado con una técnica universal. Aunque los compresores especializados tienden a ser más competitivos, las soluciones basadas en preprocesamiento son más flexibles dado que pueden adaptarse a las propiedades que caracterizan a las técnicas universales que, finalmente, se utilizan para la compresión del texto preprocesado. Esto facilita una gestión más efectiva del *trade-off* espacio/tiempo que se puede obtener con las técnicas basadas en preprocesamiento. El presente capítulo enfoca esta segunda perspectiva, cuya vertiente principal se ha centrado en la utilización de técnicas de diccionario para la transformación del texto previa a su compresión.

Como se plantea en la Sección §3.5, la compresión basada en diccionario modela el texto como una secuencia de cadenas no solapadas (*frases*) y reemplaza sus apariciones por las palabras de código que las identifican en el diccionario en el que están representadas. Por lo tanto, el concepto de *diccionario* puede entenderse como una función de mapeo del vocabulario en las palabras de código utilizadas para la representación de cada uno de sus componentes. La efectividad de este tipo de métodos radica en la obtención de una función de diccionario que permita minimizar el tamaño final del texto comprimido.

Skibiński, *et al.* [SGD05] profundizan en este tipo de técnicas, partiendo del hecho de que los vocabularios textuales tienden a estar formados por un gran número de elementos. Sobre esta premisa analizan diferentes heurísticas para la reducción del espacio necesario en la representación de estos vocabularios y plantean políticas alternativas para su organización. Este último aspecto es de vital importancia dado que la organización obtenida representa la base para la construcción de la función de diccionario.

El capítulo actual toma como punto de partida una propuesta de mapeo palabra-código en `PPM` (referida como `mPPM`) [AdlF06], cuyo enfoque planteó una perspectiva novedosa en el momento de su publicación. La propuesta actual revisa el método de construcción del diccionario considerado en `mPPM`, tanto en lo que respecta a su organización y codificación como a la gestión de las palabras contenidas, directa e indirectamente, en el vocabulario del texto. A partir de este planteamiento se obtienen diferentes heurísticas aplicables a la compresión basada en diccionario que son integradas de forma conjunta en la técnica *Word-Codeword Improved Mapping* (`WCIM`), cuya bondad se evalúa mediante la utilización de diferentes técnicas universales para la compresión del texto transformado.

En primer lugar se repasa el contexto en el que se encuadra la propuesta actual (§4.1) y a continuación (§4.2) se incide en la técnica `mPPM` que sirve como base para el presente desarrollo. La Sección §4.3 detalla las diferentes ideas que componen la propuesta actual y la forma en

las que éstas se integran en el compresor WCIM. Finalmente, la Sección §4.4 propone un estudio analítico de las propiedades de WCIM y cuantifica las mismas en un proceso de experimentación que contempla la utilización de diferentes técnicas universales de compresión. El capítulo finaliza (§4.5) con un análisis de las conclusiones extraídas sobre la propuesta actual así como de sus las posibilidades futuras de aplicación.

4.1 Trabajo relacionado

Skibiński, *et al.* introducen su trabajo [SGD05] planteando que los resultados obtenidos en compresión de texto pueden ser mejorados mediante el reemplazo de las palabras que componen el texto por referencias a las posiciones del diccionario en el que se almacenan. Además, resaltan que a pesar de la existencia de numerosas técnicas basadas en diccionario, éstas no explotan completamente las posibilidades de compresión ni consideran variantes de preprocesamiento alternativas. Esta apreciación es compartida por Abel y Teahan [AT05] cuyo trabajo, junto con el anterior, representan las revisiones más completas localizadas en del contexto actual.

Teahan y Cleary [TC96] sugieren una sencilla idea para mejorar (*boosting*) la compresión sobre un conjunto limitado de fuentes de entrada (no necesariamente textuales). Los autores plantean la posibilidad de utilizar PPM para la compresión de una determinada fuente de entrada previamente modelada con una representación que explota sus regularidades estadísticas. Esto supone que PPM inicia su proceso de compresión con el conocimiento previo aportado por el modelo construido sobre la fuente de entrada, lo cual se traduce en una mejora notable en las ratios de compresión obtenidas. Esta es, sin duda, una idea prometedora aunque plantea un problema importante: la necesidad de encontrar un modelo cuya capacidad representativa funcione bien con diferentes fuentes de entrada. En este mismo trabajo, Teahan y Cleary plantean algunas variantes para la extensión del alfabeto de entrada con q-gramas frecuentes en el texto. Los autores demuestran que extender el alfabeto con un único símbolo llega a mejorar la efectividad de PPM hasta en un 1 %.

Por otra parte, Chapin y Tate [CT98] plantean una reordenación del alfabeto de entrada que mantenga próximos, en la salida, aquellos símbolos con contextos similares. Esta propiedad favorece la compresión del mensaje de entrada con basados en la transformada *Burrows-Wheeler* (BWCA). A pesar de la originalidad de la idea, ésta sólo es aplicable a esta familia de compresores y la ganancia que proporciona está generalmente por debajo del 1 %. Abel y Teahan [AT05] examinan la interacción existente entre diferentes métodos de preprocesamiento textual y varios compresores universales. Estudian el efecto de aplicar diferentes algoritmos basados en la normalización de las letras mayúsculas, la codificación EOL, la reordenación del alfabeto (para su uso con BWCA) y, sobre todo, la mejora obtenida a través de diferentes técnicas de reemplazo de las palabras y q-gramas (*frases*) más frecuentes por nuevos símbolos que extienden el alfabeto original. Los autores justifican que el reemplazo de q-gramas sólo considera bigramas y trigramas dado que para valores mayores de q presentan una frecuencia de aparición poco relevante. En todo caso, tanto el reemplazo de palabras como el de q-gramas se lleva a cabo de forma genérica mediante un índice a la posición del diccionario en el que se almacena la cadena. La generación de los diccionario de reemplazo sobre el texto procesado permite aislar la técnica construida del idioma utilizado. Los autores consideran técnicas basadas en PPM, BWCA y LZ para la compresión del texto preprocesado. Las diferentes aportaciones planteadas en este trabajo se traducen en una mejora de las tasas de compresión comprendida entre el 3 % y el 5 % para los ficheros de texto en el corpus Calgary[1].

Los compresores derivados del algoritmo Star [KM98] representan una familia de referencia

[1] http://corpus.canterbury.ac.nz/descriptions/#calgary

en el contexto actual. Esta técnica, en su definición original, plantea un esquema de codificación basado en el reemplazo de las palabras por secuencias de símbolos '$*$' (referidos como estrellas) utilizados como referencia para el acceso a un diccionario externo organizado por longitud de palabra. La utilización de un diccionario externo, que debe ser conocido por emisor y receptor, hace que la técnica `Star` sea dependiente del idioma en el que se expresa el texto. `LIPT` [AZM+01] plantea una propuesta similar a `Star` reemplazando las secuencias de estrellas por un símbolo (del subconjunto $a \dots z$) que indica, en cada caso, la longitud de la palabra (entre 1 y 26) y por una secuencia de símbolos (en el subconjunto $a \dots z, A \dots Z$) que codifica el índice de la palabra actual dentro del conjunto de palabras de longitud l. La codificación se lleva a cabo en base 52. La mejora de esta propuesta es de un $4\,\% - 5\,\%$ respecto a `PPM` y `BCWA` y de un $7\,\%$ frente a la efectividad conseguida por `gzip`. Smirnov [Smi02] propone dos modificaciones sobre `LIPT`. Por un lado considera rangos de alfabetos disjuntos para referir las longitudes y los índices de las palabras, así como las letras de aquellas palabras no presentes en el diccionario. Esta mejora aporta una ganancia de hasta un $3\,\%$ al utilizar `bzip2` como compresor para el texto preprocesado. Por otro lado, a la modificación anterior se le añade el uso de un mayor número de subdiccionarios construidos sobre la longitud de la palabra y un etiquetado basado en *parts-of-the-speech*. Esta segunda variante mejora hasta un $2,5\,\%$ la efectividad de `LIPT` con un compresor `PPM` pero empeora la obtenida sobre `bzip2`. `StarNT` [SZM04] plantea una evolución basada en la normalización del texto. Esta técnica considera códigos de escape diferentes para señalar las palabras que empiezan por una letra mayúscula, aquellas compuestas exclusivamente por mayúsculas y, finalmente, un tercer código que indica una nueva palabra en el texto. La efectividad de esta técnica se apoya en una ordenación específica del diccionario y en el mapeo de palabras a códigos únicos. Las palabras más frecuentes se almacenan en las 312 primeras posiciones del diccionario mientras que las restantes siguen una ordenación basada en su longitud y en su frecuencia de aparición (para las palabras de la misma longitud). La técnica considera palabras de código de 1 a 3 bytes distribuidos en el rango `[a...zA...Z]` de tal forma que el diccionario tiene una capacidad máxima de 143364 entradas. La efectividad de `StarNT` mejora de un $2\,\%$ a un $5\,\%$ los resultados de `LIPT` cuando se utiliza en conjunción con `PPM` y `BCWA` y hasta un $10\,\%$ si se utiliza con `gzip`.

Skibiński, *et al.* [SGD05] plantean una revisión de las diferentes técnicas y heurísticas consideradas para preprocesamiento textual. Los autores utilizan `StarNT` como base para la evaluación de ciertas modificaciones de dicha técnica así como para la incorporación de nuevas propuestas. Entre las ideas previas, destaca cómo manejan la normalización de las mayúsculas a partir de una bandera indicativa que se incorpora justo antes de la propia palabra normalizada. Esta decisión no reduce la predictibilidad de la palabra siguiente y, a cambio, mejora la suya propia considerando que la mayoría de sus apariciones se dan tras un signo de puntuación. Por su parte, las ideas de mapeo del diccionario y de alfabetos separados (para códigos y palabras) hacen la propuesta actual completamente dependiente del idioma dado que ambas se fundamentan en que los textos en inglés tienden a estar representados con los 127 primeros símbolos del código ASCII, de tal forma los 128 símbolos finales pueden ser utilizados para propósitos de codificación. El mapeo de palabras sobre los códigos disponibles, el aumento de la capacidad total del diccionario o su reorganización (a partir de la subdivisión en intervalos del conjunto de estos códigos) y la consideración de palabras de hasta 4 bytes, representan algunas ideas novedosas de esta propuesta. Los autores consideran, también, variantes de la codificación EOL, de las políticas de reemplazo de q-gramas y del manejo de los espacios en blanco entre palabras como otras ideas de preprocesamiento. Para la compresión del texto preprocesado utilizan `PAQ6`[2],

[2] http://www.cs.fit.edu/∼mmahoney/compression/

PPMonstr[3], UHBC[4], bzip2 y gzip (la información relativa a estos compresores puede ser encontrada en el apéndice A). En su conjunto, la propuesta actual (referida como WRT) obtiene mejoras de hasta un 25 % respecto a gzip, 12 % respecto a bzip2 o un $7\,\% - 9\,\%$ para compresores como PAQ6 o PPMonstr cuya efectividad es mayor que la de las técnicas anteriores. Finalmente, este mismo trabajo trata de forma directa un escenario cuya importancia es significativa tanto en mPPM como en nuestra propuesta actual: la utilización de diccionarios de tamaño limitado en el que se descartan aquellas palabras menos frecuentes en el texto procesado. Considerando n el tamaño total del diccionario y m el número de entradas a las que éste es truncado (bajo la consideración actual), los autores evalúan el impacto derivado de la elección del límite m. Su experimentación demuestra que la pérdida de efectividad es poco significativa a partir de $m = 0,5n$. Esta conclusión demuestra, sobre su ejemplo, que la utilización de diccionarios de hasta 40000 palabras permiten obtener un *trade-off* espacio/tiempo competitivo en el entorno en el que se sitúa su estudio.

Sin embargo, las propuestas repasadas hasta ahora se centran en detalles muy específicos del texto, a excepción de esta última idea que enfoca directamente el impacto del tamaño del diccionario sobre la efectividad de las técnicas obtenidas. Éste es el punto de partida sobre el que se desarrolla la técnica mPPM [AdlF06]. Aunque en la siguiente sección se explican detalladamente sus propiedades, cabe destacar que representa el texto como una secuencia de palabras y obtiene una representación intermedia del mismo sobre un alfabeto orientado a byte ($[0, 255]$) que direcciona, mediante códigos de 2 bytes, las 2^{16} posiciones del diccionario. Esta representación es más redundante que la original y, por tanto, más compresible con la técnica PPM considerada.

mPPM, y por extensión las técnicas que se tratan a continuación, utilizan una codificación orientada a byte para representar los valores enteros que reemplazan cada aparición de una palabra por su posición en el diccionario. Las propiedades de la secuencia de bytes obtenida garantizan su compresibilidad con técnicas universales orientadas a bit. De acuerdo a lo planteado por Bell, *et al.* [BCW90], la longitud media de una palabra en textos en inglés se cifra en 5 bytes. Sin embargo la varianza es relativamente alta. Esto supone que un modelo de orden superior orientado a caracteres necesitaría un valor de k cercano a 10 para ser capaz de identificar y representar la relación existente entre dos palabras consecutivas. Sin embargo, la utilización de un modelo de palabras y su posterior codificación orientada a byte permiten obtener una longitud media de palabra de código cercana a 2 bytes y su varianza es mucho más pequeña que la obtenida para una representación basada en caracteres. Estas propiedades garantizan que los compresores basados en este tipo de transformaciones del texto son capaces de identificar correlaciones entre palabras consecutivas con un menor valor de k que las soluciones universales o c capturar correlaciones de mayor tamaño para un valor de k determinado.

Fariña *et al.* [FNP08] presentan un estudio estadístico interesante a este respecto. En él se establece la comparación entre las técnicas universales y las actualmente propuestas, y se indica la necesidad de considerar tanto las entropías de orden k como el número de contextos necesarios para alcanzarlas. Estas premisas se utilizan como base en la comparación de las propiedades de un determinado texto codificado con ETDC respecto al mismo texto analizado como una secuencia de caracteres. Los autores concluyen, de acuerdo a los resultados obtenidos, que la entropía de un texto codificado con ETDC es ligeramente menor a la del texto original para un mayor valor de k. Por lo tanto, estas técnicas requieren modelos de menor orden para conseguir una mayor efectividad que la que puede ser alcanzada al utilizar una técnica universal sobre el mismo texto. La experimentación mostrada en este trabajo demuestra el contexto teórico anterior. Para ello representan las palabras identificadas en el texto con un diccionario de tamaño ilimitado cuya organización sigue un criterio de frecuencia. Esta decisión permite (sobre las propiedades de

[3] http://www.compression.ru/ds/
[4] ftp://ftp.sac.sk/pub/sac/pack/

`ETDC`) utilizar códigos de 1 byte para la representación de las 128 palabras más frecuentes, códigos de 2 bytes para las 128^2 siguientes y así sucesivamente. Por lo tanto, el texto preprocesado se expresa con un alfabeto de salida orientado a byte: $[0, 255]$. Esta representación se comprime con `gzip`, `bzip2` y `ppmdi`, obteniendo mejoras notables respecto a la efectividad original de cada compresor. En la Sección §4.4.2 se analizan estos resultados.

Finalmente, una versión preliminar de la técnica `Edge-Guided` (completamente detallada en el capítulo siguiente) ofrece resultados comparables a los anteriores utilizando un modelado de orden 1 para la transformación del texto y manteniendo la codificación orientada a byte para la representación del texto preprocesado. Los resultados presentados en [AMPdlF07] consideran `ppmdi` como compresor final.

4.2 Propuesta de mapeo palabra-código en PPM (mPPM)

El estudio previamente comentado [SGD05] demuestra la importancia del tamaño del diccionario en el *trade-off* espacio/tiempo obtenido con una determinada técnica de compresión. El tamaño esperado para el diccionario de un texto en lenguaje natural puede aproximarse mediante la ley experimental de Heaps (§3.1). Dicha ley concluye que el crecimiento del diccionario para un determinado texto sigue una tendencia sublineal con el tamaño del propio texto. Esta propiedad aproxima el tamaño del diccionario, para textos en inglés, como $O(n^\beta)$ donde n representa el número total de palabras en el texto y $0,4 < \beta < 0,6$. Sin embargo, esta caracterización no ofrece una valoración de la relevancia de cada palabra en el texto.

La Ley de Zipf (§3.1) determina que la frecuencia relativa de la i-ésima palabra más frecuente es A_x/i^θ ($1 < \theta < 2$ para textos en inglés). Esto supone que la frecuencia de una palabra es inversamente proporcional a su ranking en la tabla general de frecuencias identificada en el texto. Por lo tanto, la distribución de frecuencia de las palabras de un texto tiene una naturaleza muy asimétrica; esto es, un pequeño conjunto de palabras posee una alta frecuencia de aparición mientras que el resto se caracteriza por una frecuencia más baja que disminuye, progresivamente, hasta un conjunto final cuyas palabras aparecen una única vez en el texto.

De la caracterización anterior puede deducirse que sólo un subconjunto del diccionario muestra una relevancia significativa dentro del texto procesado. Estas palabras deben mantenerse, de forma continuada, en el diccionario ya que tienden a ser utilizadas uniformemente a lo largo del texto (este subconjunto está formado, mayoritariamente, por artículos, preposiciones, conjunciones y algunos verbos, adverbios u adjetivos [BYRN99]). El resto de palabras, que en su conjunto representan un mayor porcentaje del diccionario que las primeras, poseen una menor significatividad dado su bajo número de ocurrencias en el texto.

`mPPM` maneja estas propiedades sobre un diccionario de tamaño limitado a 2^{16} palabras. Este tamaño es superior a la cota establecida experimentalmente por Skibiński, *et al.* [SGD05] que cifran en, al menos, 40000 palabras el tamaño a partir del cual se obtiene un *trade-off* espacio/tiempo competitivo. Esta decisión se apoya en la propia naturaleza del lenguaje natural. La capacidad del diccionario permite mantener las palabras más significativas a lo largo de todo el proceso de compresión mientras que las palabras restantes tienden a mantenerse hasta que dejan de ser utilizadas. En ese momento se eliminan del diccionario dejando espacio libre para la representación de nuevas palabras. La implementación de esta política se lleva a cabo de forma sencilla mediante un algoritmo de reemplazo `LRU` (*Least-Recently Used*). Esto es, las palabras se añaden al diccionario hasta alcanzar su capacidad máxima. Cada nueva palabra, a partir de ese momento, requiere un desalojo previo de tal forma que la posición liberada en el diccionario pasa a ser ocupada por la palabra actualmente procesada. La decisión de utilizar un reemplazo `LRU` se basa en una heurística que sostiene que las ocurrencias de las palabras poco frecuentes tienden a aparecer próximas en el texto. De esta manera, el reemplazo se lleva a cabo bajo la consideración

de que la palabra menos recientemente utilizada es aquella con una menor expectativa de volver a ser encontrada en el texto.

El tamaño del vocabulario, limitado a 2^{16} palabras, permite reemplazar cada palabra en el texto por un código de 2 bytes. A su vez, se debe considerar que `mPPM` construye su diccionario de forma progresiva de tal manera que cada nueva palabra identificada en el texto debe ser explícitamente codificada. Para ello se considera el uso de un símbolo de escape, que se añade al flujo de bytes que representa el texto, cada vez que una nueva palabra se añade al diccionario. Estos símbolos de escape permiten, a su vez, sincronizar los procesos de compresión y descompresión de forma que en este último se pueda reconstruir el diccionario utilizado en la compresión. El símbolo de escape se añade al diccionario y se identifica mediante un valor único e independiente de los utilizados para la codificación de las palabras, de forma que su interpretación, en el descompresor, no provoque ninguna ambigüedad. `mPPM` es una técnica independiente del idioma dado que no utiliza ningún diccionario previamente establecido sino que construye el suyo propio de forma completamente *adaptativa*. Finalmente, destacar que `mPPM` paraleliza el uso de dos codificadores `PPM` para la representación de su diccionario de palabras y de la secuencia de bytes que describe el texto preprocesado.

Algoritmo 4.1 Compresión mPPM

```
1: Σ ← λ;
2:
3: for all p ∈ T do
4:    if p ∈ Σ then
5:       codifica(PPM_text, p)
6:    else
7:       if σ = 2^16 then
8:          p' ← Σ.lru()
9:          Σ ← Σ − p';
10:      end if
11:
12:      Σ ← Σ ∪ p;
13:
14:      codifica(PPM_text, λ)
15:      for all c ∈ p do
16:         codifica(PPM_voc, c)
17:      end for
18:   end if
19: end for
```

El algoritmo 4.1 muestra la descripción original de la técnica de compresión `mPPM`. Σ se utiliza como referencia al vocabulario de palabras sobre el que se construye la función de diccionario. Σ se declara como un *conjunto ordenado* formado por σ elementos. El símbolo λ representa el valor de escape utilizado para la codificación de una nueva palabra. PPM_{text} y PPM_{dicc} son los codificadores `PPM` utilizados en la compresión, respectivamente, de la secuencia de bytes que describe la representación del texto preprocesado y de las palabras añadidas, carácter a carácter, a Σ. Finalmente, p y p' son, respectivamente, la palabra actualmente procesada y la seleccionada para desalojo si $\sigma = 2^{16}$. `mPPM` procesa el texto en conjunción con la transformación *spaceless words* [dMNZ97].

Skibiński, *et al.* [SGD05] plantean las ventajas e inconvenientes de utilizar métodos basados en el reemplazo de palabras para la compresión de texto. En primer lugar, señalan que el concepto

de reemplazo de palabras por códigos de menor longitud, sobre una determinada función de diccionario, tiene al menos dos debilidades:

1. El diccionario tiene un gran tamaño (con capacidad para decenas de miles de palabras) apropiado para lenguaje natural.

2. Los métodos basados en diccionario ofrecen una representación del texto de orden 0 lo cual limita la posibilidad de detectar y representar las relaciones de alto nivel que existen entre las palabras.

Sin embargo, también destacan el beneficio de estas técnicas ya que, dada su sencillez, desarrollan una alta velocidad en la construcción del diccionario y el procesamiento del texto, obteniendo unas tasas de compresión competitivas.

La orientación a diccionario que sigue `mPPM` le permite aprovechar estas propiedades y, sobre ellas, afrontar las debilidades anteriores. El diccionario resultante, con capacidad para 2^{16} palabras, se sitúa en el tamaño esperado de varias decenas de miles de palabras. Aún así, es un diccionario fácilmente tratable con los recursos de memoria actualmente disponibles en un ordenador convencional y, por lo tanto, no compromete la eficiencia de la técnica. Además, debe notarse como la limitación establecida reduce notablemente el porcentaje de palabras almacenadas en comparación con el tamaño total esperado para textos de mediano y gran tamaño. Por otro lado, aunque la técnica de mapeo palabra-código utilizada por `mPPM` no es capaz de aprovechar directamente las relaciones de alto nivel existentes entre las palabras, sí facilita su identificación por parte de la técnica de orden superior (`ppmdi`) utilizada para la compresión del texto preprocesado. De acuerdo a las conclusiones planteadas por Fariña, *et al.* [FNP08], la representación orientada a byte obtenida por `mPPM` permite que un compresor de orden superior identifique y aproveche las relaciones existentes entre palabras con un menor valor de k. Un compresor orientado a caracteres necesita un valor de k cercano a 10 para ser capaz de identificar la relación entre dos palabras consecutivas, mientras que sobre `mPPM` esta misma relación puede ser obtenida con $k = 4$, considerando que cada palabra se representa mediante códigos de 2 bytes. La experimentación presentada en [AdlF06] contempla que la mejora de `mPPM`, respecto a `ppmdi`, alcanza hasta un 14 % para textos de mediano y gran tamaño. Finalmente, es importante resaltar que `mPPM` obtiene tiempos de compresión/descompresión inferiores a los de `ppmdi`.

4.3 WCIM (Word-Codeword Improved Mapping)

La experiencia previa obtenida con la técnica `mPPM`, junto con el conocimiento derivado del estudio mostrado en [FNP08] y las leyes experimentales que caracterizan el lenguaje natural sugieren la posibilidad de mejorar los resultados obtenidos por la técnica anterior. *Word-Codeword Improved Mapping* (`WCIM`) plantea una técnica *semi-estática* a partir de la revisión del mapeo palabra-código utilizado en `mPPM`.

La ley de Zipf sugiere una distribución muy asimétrica sobre las frecuencias de aparición de las palabras del texto. Esto supone, como se comentaba previamente, que un pequeño subconjunto de palabras acapara un elevado número de ocurrencias en el texto. Estas palabras son especialmente *significativas* de acuerdo con su alta frecuencia de aparición. Por lo tanto, siguiendo el principio fundamental de la compresión, la codificación de estos símbolos debe ser tratada de forma efectiva mediante palabras de código de menor longitud. Esta decisión mejora el tratamiento que estas palabras reciben en `mPPM`, dónde todas las palabras se codifican con 2 bytes. Las secciones §4.3.1 y §4.3.2 detallan como la nueva técnica gestiona este aspecto y el impacto que supone en el tamaño final del diccionario utilizado en el caso actual.

El conjunto de palabras restantes muestra una frecuencia de aparición decreciente de forma inversamente proporcional a su ranking en el diccionario hasta alcanzar un subconjunto final con una única ocurrencia en el texto. Esto nos permite subdividir y categorizar este conjunto de palabras. Por un lado se plantea una gestión específica para aquellas palabras ("*unitarias*") cuya única aparición en el texto puede ser representada sin necesidad de introducir la palabra en el diccionario (§4.3.4). Las palabras restantes, calificadas como *poco frecuentes*, se representan en el diccionario mediante una nueva heurística (Sección §4.3.3) basada en la utilización de una lista de espera que mejora la efectividad del reemplazo de palabras llevado a cabo en `mPPM`.

La primera pasada de `WCIM` obtiene una caracterización estadística del diccionario. Para cada palabra almacena tanto su número total de ocurrencias como una marca de tiempo que identifica la primera de ellas. Esta información se utiliza para la construcción y organización del diccionario de acuerdo a las siguientes heurísticas. Una vez obtenida esta configuración, se produce el reemplazo de cada palabra en el texto por su identificador en el diccionario. Este resultado se comprime, finalmente, con una técnica universal `X`, que da lugar a diversos compresores $\texttt{WCIM}_X$ cuyas propiedades se analizan en la sección de experimentación.

4.3.1. Promoción de las palabras más significativas

La efectividad de una técnica de compresión radica, fundamentalmente, en la utilización de un menor número de bits para la codificación de los símbolos más frecuentes. Sin embargo, el planteamiento original de `mPPM` utiliza palabras de código de longitud 2 bytes para representar todos los símbolos.

`WCIM` utiliza las estadísticas obtenidas en la primera pasada para la ordenación (basada en frecuencia) del diccionario. Esta decisión permite identificar como *significativas* las 128 palabras más frecuentes. De acuerdo a esta caracterización, `WCIM` opta por una codificación de longitud variable orientada a byte. Las palabras *significativas* se promocionan para su representación con palabras de código de un sólo byte. El resto de palabras, en el diccionario, se representan (al igual que en `mPPM`, usando palabras de código de 2 bytes. Por lo tanto, el esquema de codificación de `WCIM` diferencia palabras de código de 1 y 2 bytes mediante el valor del primer bit del primer byte de la palabra de código. Si el valor de este primer bit es `0`, la palabra de código está formada únicamente por el byte actual. Si, por el contrario, el primer bit es `1`, entonces la palabra de código comprende tanto el byte actual como el siguiente. Nótese como el segundo byte no reserva su primer bit para propósitos de marcado ya que la información aportada por el primero es suficiente para identificar el tipo de palabra de código. Este hecho nos permite disponer de un rango de 2^{15} palabras de código de longitud 2 bytes respecto a las 2^{14} que se obtendrían con una codificación `ETDC` pura. Esta decisión da lugar a la existencia de dos rangos de codificación: de `<0`0000000`>` a `<0`1111111`>` para las palabras significativas y de `<1`0000000 00000000`>` a `<1`11111111 11111111`>` para el resto de palabras en el diccionario principal. Esta modificación en el esquema de codificación influye en la capacidad de almacenamiento del diccionario actual.

4.3.2. Reducción del tamaño total del diccionario

Promocionar la codificación de las palabras más significativas reduce la capacidad del diccionario utilizado en `WCIM`. Reservar el primer bit de cada byte, para propósitos de marcado, reduce el número de posiciones direccionables en el diccionario. Mientras que `mPPM` podía referenciar hasta 2^{16} posiciones, el diccionario utilizado en `WCIM` puede almacenar hasta $2^7 + 2^{15}$ palabras diferentes. Esto supone que el diccionario actual tiene, aproximadamente, la mitad de la capacidad del anterior. Sin embargo, el tamaño actual aún se mantiene próximo al indicado como referencia en el estudio de Skibiński, *et al.* [SGD05] y, atendiendo a la ley de Zipf, dispone de una capacidad suficiente como para representar las palabras más significativas del texto durante todo

el proceso de compresión y soportar el reemplazo de aquellas otras cuyo utilización evoluciona en el texto. Como se planteaba en el estudio de `mPPM`, las palabras poco frecuentes tienden a aparecer próximas en el texto y `WCIM` dispone de mayor información para su gestión y reemplazo gracias a su modelado semi-estático.

El conjunto de palabras diferentes (Σ) se ordena de acuerdo a su frecuencia de aparición. Para aquellos subconjuntos caracterizados por el mismo número de ocurrencias se utiliza un segundo criterio de ordenación basado en la marca de tiempo que referencia su primera aparición en el texto. De esta manera, cada subconjunto de palabras con la misma frecuencia está internamente ordenado de acuerdo a su primera ocurrencia, priorizando la representación de aquellas palabras que aparecen con anterioridad en el texto. El diccionario de `WCIM` se construye atendiendo a estas propiedades. Las $2^7 + 2^{15}$ primeras palabras se representan directamente en el diccionario principal mientras que las restantes se gestionan de forma independiente. Esto significa que una gran cantidad de las palabras poco frecuentes, y todas las "unitarias", no se representan inicialmente en el diccionario para texto de mediano y gran tamaño. Aún así, el subconjunto de palabras poco frecuentes representadas facilita la implementación de un reemplazo efectivo.

`WCIM` desarrolla su política de reemplazo sobre la misma propiedad considerada en `mPPM`: la proximidad que presentan en el texto las ocurrencias de las palabras poco frecuentes. Estas palabras tienden a ser desalojadas del diccionario principal antes de que las no representadas sean identificadas en el texto. Esta propiedad se aproxima a través de la marca temporal y de la heurística `LRU` utilizada en el reemplazo. El contador de ocurrencias de una palabra se decrementa en una unidad cada vez que ésta se encuentra en el texto (durante el proceso de compresión). De esta manera, la palabra se desaloja del diccionario cuando su contador de ocurrencias alcanza el valor 0, garantizando que no volverá a ser encontrada en el texto. La posición liberada pasa a ser ocupada, como se plantea a continuación, por aquella palabra no representada en el diccionario cuya aparición en el texto sea más próxima de acuerdo al estado actual del proceso.

4.3.3. Utilización de una lista de espera para palabras poco frecuentes

El número de palabras poco frecuentes tratadas en el caso actual representa una fracción importante de Σ (nótese que las palabras "unitarias" tienen un tratamiento específico e independiente del explicado en el caso actual). La representación y gestión de este subconjunto de palabras se lleva a cabo mediante un segundo diccionario (denominado *lista de espera*) cuyo tamaño es ilimitado y está directamente vinculado con el diccionario principal. La lista de espera se ordena de acuerdo al número de ocurrencias de cada palabra y cada subgrupo con la misma frecuencia sigue el orden establecido por la marca de tiempo de la primera ocurrencia de cada una de las palabras que lo forman.

La posición liberada en cada reemplazo del diccionario principal pasa a ser ocupada por la primera palabra en la lista de espera. Esta decisión permite actualizar el diccionario principal con aquella palabra cuyas ocurrencias se preveen más próximas de acuerdo al estado actual del proceso de compresión.

Al igual que para las palabras representadas en el diccionario principal, el contador de ocurrencias de una palabra almacenada en la lista de espera se decrementa en una unidad cada vez que ésta se encuentra en el texto. Sin embargo, a diferencia del caso anterior, si dicho contador alcanza el valor `0`, la palabra almacenada en la posición j, se elimina de la lista de espera. La codificación de una palabra identificada en la lista de espera requiere la utilización de un símbolo de escape $\mathtt{W}_{lista}$ que antecede a la palabra de código utilizada propiamente para su representación. Este símbolo de escape se representa como una palabra virtual en el diccionario principal y se codifica de acuerdo a su posición en el mismo. Las estadísticas de esta palabra recogen como frecuencia el número total de ocurrencias de las palabras de la lista de espera y como marca

de tiempo utiliza la asociada a la palabra inicialmente almacenada en la primera posición de la lista de espera.

La codificación de las palabras en la lista de espera se construye directamente sobre `ETDC`. Esto supone un rango de 128 códigos de 1 byte para las primeras posiciones de la lista, 128^2 códigos de 2 bytes y así sucesivamente. A pesar del tamaño ilimitado de la lista de espera, la mayoría de las codificaciones en la misma se van a realizar sobre palabras almacenadas en su parte inicial. Esto supone que las palabras almacenadas en las 128 primeras posiciones de la lista de espera se siguen codificando con 2 bytes dado que, en términos prácticos, la codificación de $\mathtt{W}_{lista}$ requiere un único byte.

4.3.4. Gestión basada en cola para palabras "unitarias"

Finalmente se explica el tratamiento de las palabras "unitarias', entendiendo como tal aquellas que aparecen una única vez en el texto. La codificación de estas palabras no requiere de un almacenamiento previo en el diccionario. Su única ocurrencia se representa por la propia palabra y no por una referencia a ella. Para este propósito se considera una tercera estructura de almacenamiento denominada *cola de "unitarias"*. Dicha estructura se construye como una cola cuya ordenación se lleva a cabo de acuerdo a la marca de tiempo de cada palabra "unitaria", de manera que las primeras en aparecer en el texto se sitúan en las primeras posiciones.

La codificación de las palabras "unitarias" requiera le definición de una segunda palabra virtual denominada W_{unit}. Al igual que la anterior, esta palabra se gestiona en el diccionario principal y su frecuencia de aparición se calcula como el número de palabras "unitarias" identificadas en el texto mientras que su marca de tiempo representa la primera ocurrencia de una de estas palabras. El procedimiento de codificación es muy sencillo: cada ocurrencia de una palabra "unitaria" en el texto se reemplaza por el código que identifica W_{unit} en el diccionario principal. La estructura se actualiza mediante una operación `pop`, de tal forma que la primera palabra en la cola es desalojada.

4.3.5. Implementación

La implementación de `WCIM` integra las cuatro políticas anteriores. El proceso de desarrollo se lleva a cabo en lenguaje `C++` utilizando los recursos ofrecidos por la *Standard Template Library*[5] (`STL`) para la implementación de algunas estructuras de datos y algoritmos.

El algoritmo 4.2 muestra una descripción funcional del proceso de compresión `WCIM`. La pasada inicial (primer bucle `for`) procesa el texto ($\mathcal{T}$), de acuerdo a la transformación *spaceless words*, obteniendo el vocabulario Σ y la caracterización estadística de cada una de sus palabras (esta información se acumula mediante las invocaciones al método `actualizar`). Las estadísticas extraídas se utilizan para la ordenación final de Σ de la que se obtienen el diccionario principal, la lista de espera y la cola de "unitarias". Esta operación tiene un coste $\Theta(\sigma \log \sigma)$ sobre una ordenación tipo *quicksort* [Hoa62] implementada mediante el algoritmo `sort` de la `STL`. Las palabras virtuales $\mathtt{W}_{lista}$ y $\mathtt{W}_{unit}$ se añaden (líneas 12 y 13) a Σ de acuerdo a las estadísticas que caracterizan tanto la lista de espera como la cola de "unitarias".

En el punto actual del algoritmo (línea 14) se dispone ya de la configuración final del diccionario (repartida en las tres estructuras planteadas). El método `codifica` implementa el proceso de codificación de Σ a través de un compresor `PPM` en el que cada palabra se representa, carácter a carácter, bajo las mismas propiedades consideradas en `mPPM`.

La implementación de Σ se basa en una *tabla hash* que almacena todas las palabras. Dicha estructura se obtiene a partir de una adaptación del código propuesto por Justin Zobel[6] [ZWH01].

[5] http://www.sgi.com/tech/stl/

[6] http://www.cs.mu.oz.au/~jz/resources/index.html

Algoritmo 4.2 Compresión WCIM

```
1: Σ ← ∅;
2:
3: for all p ∈ T do
4:    if p ∈ Σ then
5:       actualizar(Σ, p)
6:    else
7:       Σ ← Σ ∪ p;
8:    end if
9: end for
10:
11: Σ.ordenar();
12: Σ ← Σ ∪ W_lista;
13: Σ ← Σ ∪ W_unit;
14: codifica(PPM_voc, Σ)
15:
16: for all p ∈ T do
17:    cod ← info(Σ, p)
18:
19:    if cod = UNIT then
20:       vcod ← info(Σ, W_unit)
21:       codifica(Coder, vcod)
22:
23:       Σ ← Σ − cod
24:    else
25:       if cod ≥ 2^7 + 2^15 then
26:          vcod ← info(Σ, W_lista)
27:          codifica(Coder, vcod)
28:          codifica(Coder, cod)
29:       else
30:          codifica(Coder, cod)
31:       end if
32:
33:       occs ← procesar(Σ, cod)
34:       if occs = 0 then
35:          Σ ← Σ − cod
36:       end if
37:    end if
38: end for
```

Cada elemento en la tabla almacena el identificador que referencia la posición de la palabra en Σ, a excepción de las palabras "unitarias" que se identifican con la constante `UNIT` (considerando que se acceden de forma ordenada a través de la pila). Tanto el diccionario principal como la lista de espera se implementan sobre vectores `STL` en los que cada posición almacena las estadísticas asociadas a la palabra representada en ella. De la misma forma, la pila se implementa sobre un tercer vector `STL` que facilita la implementación de la operación `pop` con el método `erase` y el uso de iteradores. El coste espacial de estas estructuras tiene un orden $O(\sigma)$ y facilita obtener una implementación eficiente para las operaciones de compresión requeridas.

La segunda pasada sobre $\mathcal{T}$ ejecuta el proceso de compresión sobre la configuración de Σ previamente obtenida. El texto se procesa de forma similar a la primera pasada y cada palabra se obtiene sobre la misma trasformación *spaceless words*. Recuperar el identificador de la palabra en el diccionario tienen un coste $O(1)$ sobre la tabla hash. Si este valor es igual a `UNIT` (línea 19) la palabra procesada es la primera en la cola de "unitarias". El método `codifica` implementa la representación de cada código sobre la técnica de compresión universal (`Coder`) elegida en cada caso. Una vez realizada la codificación de W_{unit}, se ejecuta la función `pop` (línea 23) sobre la cola de tal forma que la nueva primera palabra representa la próxima "unitaria" que será procesada en el texto.

Si, por el contrario, la palabra codificada no es "unitaria" se puede deducir fácilmente si está representada en el diccionario principal o en la lista de espera. Si su identificador es mayor o igual a $2^7 + 2^{15}$, la palabra está almacenada en la lista de espera (línea 25). En primer lugar se codifica W_{lista} y, a continuación, se representa el código de la palabra en la lista de espera. Se accede a Σ para decrementar en una unidad el contador de ocurrencias de p mediante el método `procesar`. Si su contador es igual a 0, la palabra se desaloja de la lista de espera (línea 35).

Finalmente, si el identificador `cod` de p es inferior a 2^7+2^{15}, nos encontramos ante una palabra representada en el diccionario principal (línea 30). La codificación de esta palabra representa, directamente, su identificador `cod` mediante la función `codifica` utilizada en los casos anteriores. Al igual que para palabras en la lista de espera, se actualiza la información estadística de la palabra de forma que si ésta es su última ocurrencia, entonces la palabra se desaloja. Esto conlleva la necesidad de promocionar la primera palabra en la lista de espera que pasa a ocupar la posición liberada por p en el diccionario principal.

La información, almacenada en la tabla hash, acerca de la composición de la lista de espera puede quedar desactualizada tras las operaciones de desalojo. Esto conlleva un paso previo de comprobación cada vez que una de estas palabras va a ser codificada. Este paso consiste en comprobar que el identificador indicado en la tabla corresponde con la posición de la lista de espera. De no ser así precisaría localizar su nueva ubicación. El coste de esta operación es lineal en el rango comprendido entre las posiciones anterior y actual de la palabra.

La figura 4.1 muestra una descripción conceptual del proceso de compresión con `WCIM`. Por un lado se muestran el diccionario principal y la lista de espera, junto a la cola de "unitarias". La figura *texto preprocesado* representa una secuencia de códigos enteros obtenidos como resultado del preprocesamiento obtenido con `WCIM`. La primera posición en la secuencia referencia la palabra almacenada en la posición 127 del diccionario, mientras que la segunda apunta a la i-ésima posición en el mismo que está ocupada por la palabra virtual W_{unit}. Esto le indica al descompresor que tiene que sustituir el código actual por la primera palabra almacenada en la cola de "unitarias" y, posteriormente, ejecutar una función `pop` sobre ella. El tercer código 1, indica que la siguiente palabra se debe buscar en la lista de espera, mientras que el siguiente código (2) resuelve que la palabra a añadir en descompresión está en la segunda posición de la lista de espera. Finalmente, esta secuencia de enteros se comprime con una técnica universal (representada por `Coder` en el algoritmo 4.2). En la figura se muestran las diferentes posibilidades de compresión consideradas en la siguiente sección de experimentación.

4.4 Experimentación

En la presente sección se detallan los resultados obtenidos sobre las heurísticas implementadas en la nueva técnica `WCIM`. En primer lugar se analizan sus propiedades desde la perspectiva analítica que sustenta su diseño. Este estudio afronta el análisis de las propiedades de los diccionarios obtenidos con `WCIM` y los compara respecto a los utilizados en `mPPM`. Finalmente, se evalúa la efectividad de `WCIM` sobre un corpus heterogéneo de textos y se compara respecto a otras

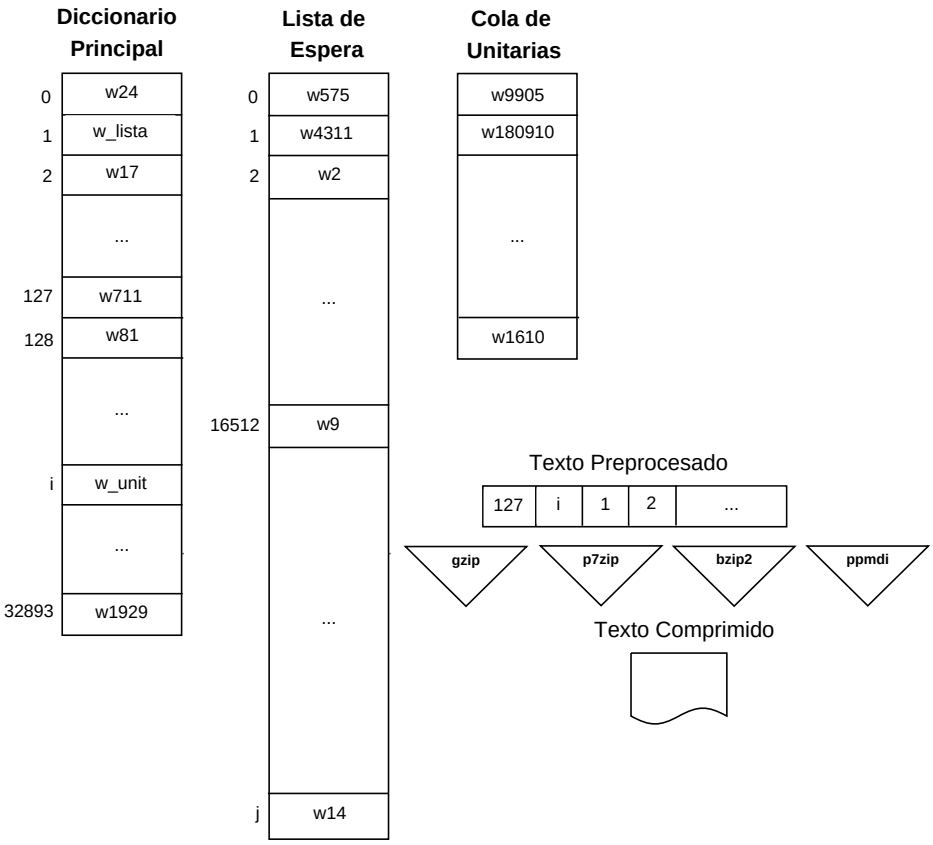


Figura 4.1: Proceso de compresión con WCIM sobre la configuración final del diccionario.

propuestas de compresión, entre las que se incluye `mPPM` y otras técnicas comparables. En lo que respecta a `mPPM`, se extiende su análisis original (centrado exclusivamente en `ppmdi`) con el fin de evaluar y comparar sus propiedades con otras técnicas de compresión universal. La descripción de los recursos utilizados en este proceso de experimentación están completamente detallados en el apéndice A.

4.4.1. Estudio analítico

El estudio actual se desarrolla bajo las condiciones planteadas en [FNP08] con la finalidad de evaluar cómo el preprocesamiento orientado a byte obtenido por `WCIM` es aún compresible con una técnica universal orientada a bit. También se analizan las propiedades del preprocesamiento obtenido con `mPPM` con el objetivo de estimar la mejora alcanzada con la técnica actual. Nos centraremos en el análisis de las entropías de orden k (H_k) tanto del texto original como de su transformación sobre con el preprocesamiento obtenido a través de `WCIM` y `mPPM`.

La tabla 4.1 muestra las entropías de orden k obtenidas para la colección `FT94` así como el número de contextos necesarios para conseguir estos valores[7]. Estos cálculos consideran un modelado orientado a byte (por lo tanto sobre un alfabeto de salida de 256 elementos: [0,255]) tanto del texto plano como de sus respectivas transformaciones obtenidas sobre el preprocesamiento descrito por `mPPM` y `WCIM`. Al procesar el texto como una secuencia de bytes, el modelado de orden k almacena las estadísticas de cada carácter c_i de acuerdo a los k caracteres que lo preceden.

[7]Los valores H_k se calculan utilizando software disponible en `http://pizzachili.dcc.uchile.cl/`.

k	Plano		mPPM		WCIM	
	H_k	Contextos	H_k	Contextos	H_k	Contextos
0	5,0006	1	6,0759	1	7,1936	1
1	3,6388	88	5,2173	256	6,1731	256
2	2,7840	5.113	3,8102	65.536	4,5879	61.768
3	2,1166	74.498	2,8322	2.931.424	3,0036	4.100.557
4	1,6963	524.699	2,0413	9.846.573	1,6048	18.174.060
5	1,4611	2.072.515	1,4403	21.018.203	0,7718	32.918.692
6	1,3078	5.420.628	0,9492	33.199.805	0,3954	41.941.335
7	1,1735	10.943.311	0,5777	44.052.423	0,2332	46.648.654
8	1,0417	18.788.845	0,3672	52.087.305	0,1602	49.121.812
9	0,9120	28.878.013	0,2295	57.590.425	0,1230	50.521.715
10	0,7893	40.693.718	0,1715	60.990.170	0,1059	51.398.708
20	0,1285	141.687.067	0,0511	68.954.324	0,0466	55.235.167
30	0,0392	164.993.841	0,0357	71.243.443	0,0316	57.561.360
40	0,0242	171.102.511	0,0289	72.948.916	0,0156	59.347.598
50	0,0195	174.979.550	0,0266	74.476.971	0,0087	60.687.965

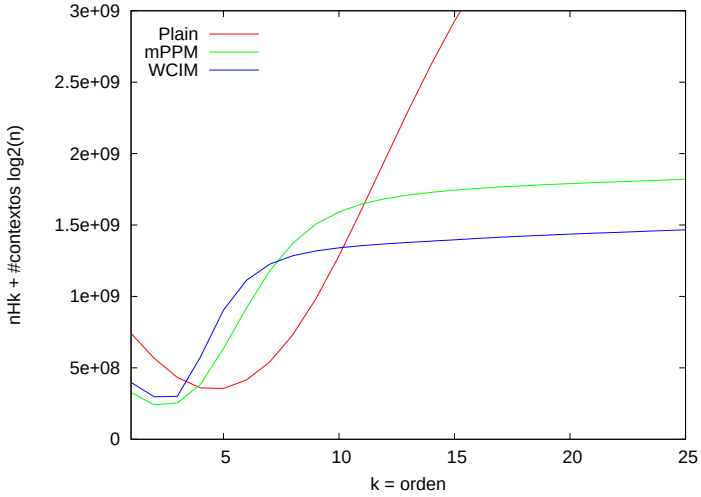


Tabla 4.1: Estudio de las entropías de orden k para la colección FT94 y valores $nHk + \#contextos\ log_2(n)$ obtenidos para el mismo.

Esto supone que las representaciones obtenidas para valores menores de k poseen una capacidad inferior para la identificación de correlaciones entre caracteres y por tanto su efectividad tiende a ser menor. Sin embargo, al aumentar el orden del modelo se consigue una mejor representación de las estadísticas del texto [BCW90] a costa de aumentar el número de contextos utilizados, lo cual puede transformar el modelo en una solución impracticable en tiempo y espacio.

La longitud media de una palabra en inglés es, aproximadamente, 5 bytes [BCW90], aunque su varianza es relativamente alta. Esto supone, en general, que un modelo de orden superior orientado a caracteres necesita un valor de k cercano a 10 para ser capaz de identificar y aprovechar la relación existente entre dos palabras consecutivas en el texto. Desde esta perspectiva, el preprocesamiento del texto llevado a cabo tanto por mPPM como por WCIM supone una ventaja considerable. Aún considerando que ambas propuestas utilizan el byte como unidad mínima de representación, la longitud media de una palabra de código en mPPM es 2 bytes mientras que en WCIM tiende a ser un valor ligeramente inferior. En lo que respecta a su varianza, mPPM obtiene un valor 0 dado que representa todas las palabras con códigos de longitud 2 bytes. Por su parte, la varianza de las representaciones obtenidas por WCIM tiende a ser un valor mucho más pequeño que el obtenido al modelar el texto plano dado que las palabras se representarán con códigos de 1 o 2 bytes, excepto en aquellos casos en los que se trate de una palabra almacenada en la lista de espera a partir de su posición 128. En este caso, se necesitarán 3 bytes para su representación.

Estas propiedades garantizan que, sobre el texto preprocesado, un modelo de orden superior es capaz de identificar transiciones entre palabras consecutivas con un valor menor de k o capturar correlaciones mayores para un valor de k determinado.

Sin embargo, esta comparación no puede llevarse, a cabo de forma exclusiva, sobre las propiedades anteriores dado que un compresor de orden k requiere manejar y representar la información asociada al modelo utilizado. Por lo tanto, una comparación correcta entre las propiedades de las representaciones obtenidas sobre el texto "en plano" y sus transformaciones basadas en preprocesamiento debe considerar tanto los valores de entropía (obtenidos para un valor de k dado) como el número de *contextos* necesarios para ello. Por ejemplo, la tabla 4.1 muestra que la entropía de orden 6 para texto plano precisa representar casi 5,5 millones de contextos, mientras que un número similar de contextos son los necesarios para alcanzar un valor intermedio entre H_3 y H_4 para las transformaciones obtenidas con `mPPM` y `WCIM`. Pero los valores de H_k no son directamente comparables dado que la utilización de un compresor de orden k, cómo se decía anteriormente, está gravada por la necesidad de almacenar y manejar un número determinado de contextos. Esto no sólo influye en la cantidad de memoria necesaria sino también en la necesidad de codificar dichos contextos. La figura posicionada bajo la tabla anterior muestra la comparativa obtenida para el estudio propuesto considerando que la representación de cada contexto se penaliza con $log_2(n)$ bits necesarios para su codificación sobre el resultado final.

En dicha figura puede observarse como los mínimos de `mPPM` y `WCIM` son inferiores al obtenido para el texto plano. Además, este valor se consigue con un menor valor de k lo que promociona ambas técnicas respecto a una compresión universal del texto sobre un modelo de orden superior. La curva descrita por `mPPM` es ligeramente mejor a la de `WCIM` hasta $k = 7$ (a partir de este valor `WCIM` es notablemente más efectiva).

Este resultado está relacionado con la codificación de las palabras más significativas. Mientras que `WCIM` promociona las 128 primeras palabras (que son codificadas con un único byte), `mPPM` codifica todo su diccionario con palabras de código de 2 bytes. Esto supone que las 256 palabras más frecuentes comparten su primer byte: `<00000000>`. La figura 4.2 muestra la distribución de frecuencia de los bytes obtenida de acuerdo al preprocesamiento de la colección `FT94` sobre `mPPM` y `WCIM`. El eje X representa (en escala logarítmica) los 256 elementos, en el alfabeto de salida, siguiendo un orden creciente de frecuencia. El primer byte en `mPPM` tiene una frecuencia de 26,37 % mientras que la frecuencia del primer byte en `WCIM` es 7,97 %. Este hecho es el que motiva la tendencia de `mPPM` respecto a `WCIM` y tiene una influencia notable en la compresibilidad de la secuencia.

Sin embargo, aunque la curva de `mPPM` alcanza un valor mínimo inferior al de `WCIM`, la efectividad de la primera técnica no depende únicamente de la secuencia de bytes obtenida. La técnica de reemplazo llevada a cabo en `mPPM` degrada el resultado obtenido en la codificación del diccionario dado que una misma palabra puede ser representada en diferentes ocasiones (tantas como entre al diccionario después de haber sido desalojada). Esta propiedad supone que `mPPM` consigue una mejor efectividad que `WCIM` para aquellos casos en los que el coste de recodificar las palabras desalojadas sea inferior al beneficio obtenido sobre las heurísticas contempladas en `WCIM`. Cómo se muestra a continuación, esto hace que `mPPM` sea la propuesta más competitiva para textos de menor tamaño, pero su comportamiento se deteriora para colecciones de gran tamaño en los que la técnica de reemplazo se hace necesaria para el modelado del texto. Esta propiedad se estudia a través de los datos planteados en la tabla 4.2.

Dicha tabla detalla las propiedades estadísticas de los diccionarios obtenidos con `mPPM` sobre textos de diferente tamaño (colecciones `WSJ`, `CR` y `FT94`) con los que disponer de un corpus de estudio más heterogéneo. La columna σ muestra el número de palabras diferentes identificadadas en el texto (tamaño del vocabulario), "Reemps." es el número de palabras reemplazadas y "Cods. mPPM" representa el número total de palabras codificadas; esto es, el número de palabras

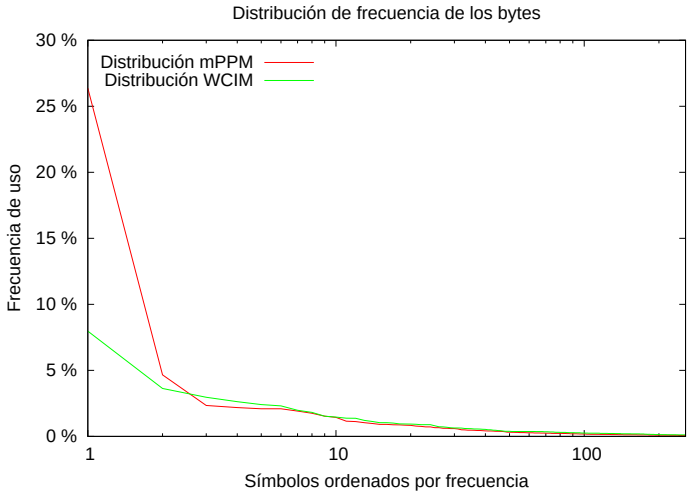


Figura 4.2: Distribuciones de frecuencia de los bytes utilizados en el preprocesamiento obtenido por mPPM y WCIM sobre el texto FT94.

Texto	Palabras			
	σ	Reemps.	Cods. mPPM	
WSJ001	19352	0	19352	0 %
WSJ010	54542	0	54542	0 %
WSJ100	157381	169408	234941	45,78 %
FT94	295018	555740	621273	58,71 %
CR	117713	73705	139238	29,20 %

Tabla 4.2: Caracterización estadística de los diccionarios obtenidos con mPPM.

diferentes en el texto más las palabras que entran de nuevo en el diccionario después de haber sido desalojadas. La columna final muestra el incremento porcentual del número de palabras codificadas en mPPM respecto al tamaño del vocabulario. Como puede apreciarse, para los ficheros de pequeño tamaño no se completa la capacidad del diccionario. Esto supone que no se lleva a cabo ningún reemplazo y, por tanto, el vocabulario codificado es exclusivamente el formado por las σ palabras diferentes identificadas (sin la codificación repetida de ninguna de ellas). Sin embargo, para el resto de textos, se observa como el incremento del número de palabras codificadas está entre el $29,20\,\%$ y $58,71\,\%$ lo que supone una importante sobrecarga en el tamaño final del texto preprocesado con mPPM. Esta propiedad conlleva que la técnica mPPM sea, actualmente, una opción menos competitiva para textos de gran tamaño. En la siguiente sección se cuantifican estas propiedades dentro de un entorno de experimentación real en el que se consideran diferentes técnicas para la compresión del texto preprocesado.

4.4.2. Resultados experimentales

La presente sección plantea un resumen detallado de los resultados experimentales obtenidos tanto por WCIM como con mPPM, cuyo estudio se extiende sobre otras técnicas universales complementarias a la considerada en el trabajo original [AdlF06]. Nótese que los resultados obtenidos para $\texttt{WCIM}_{ppmdi}$ se corresponden con los planteados en [AMPdlF09] bajo la referencia mPPM-2.

Estos resultados se comparan respecto a otras técnicas basadas en la combinación de un preprocesamiento del texto orientado a byte y su compresión con una técnica universal. Por un lado, se considera la técnica ETDC+X [FNP08] que preprocesa el texto sobre las propiedades de ETDC y, posteriormente, le aplica un compresor universal: X. Por otro lado, se utiliza el compresor Edge-Guided [AMPdlF07] que preprocesa el texto sobre un modelo de orden 1 y

Texto	ETDC	E-G	mPPM	WCIM
AP001	44,21 %	49,03 %	55,41 %	**44,20 %**
AP005	37,64 %	40,43 %	48,26 %	**37,21 %**
AP010	36,15 %	38,27 %	46,65 %	**35,60 %**
AP020	34,88 %	36,21 %	45,35 %	**34,34 %**
AP040	33,99 %	34,72 %	44,61 %	**33,56 %**
AP060	33,65 %	34,06 %	44,40 %	**33,30 %**
AP100	33,34 %	33,40 %	44,23 %	**33,07 %**
WSJ001	43,98 %	48,88 %	55,07 %	**43,97 %**
WSJ005	37,77 %	40,85 %	48,40 %	**37,37 %**
WSJ010	36,12 %	38,39 %	46,69 %	**35,62 %**
WSJ020	34,81 %	36,28 %	45,27 %	**34,29 %**
WSJ040	33,89 %	34,66 %	44,55 %	**33,48 %**
WSJ060	33,57 %	34,00 %	44,39 %	**33,23 %**
WSJ100	33,23 %	33,05 %	44,24 %	**32,95 %**
ZIFF001	**40,80 %**	43,79 %	53,45 %	41,08 %
ZIFF005	36,95 %	38,84 %	48,53 %	**36,70 %**
ZIFF010	36,09 %	37,58 %	47,44 %	**35,66 %**
ZIFF020	35,27 %	36,20 %	46,56 %	**34,79 %**
ZIFF040	34,53 %	34,91 %	45,97 %	**34,12 %**
ZIFF060	34,22 %	34,30 %	45,81 %	**33,90 %**
ZIFF100	33,97 %	**33,72 %**	45,76 %	33,73 %
FT91	35,53 %	38,07 %	45,91 %	**34,82 %**
FT92	32,82 %	32,71 %	44,58 %	**32,57 %**
FT93	32,87 %	**32,15 %**	45,03 %	32,63 %
FT94	32,83 %	**32,09 %**	44,93 %	32,59 %
CR	31,94 %	31,76 %	41,78 %	**31,57 %**

Tabla 4.3: Efectividad del preprocesamiento de WCIM y otras técnicas comparables.

codifica el resultado con ETDC como paso previo a su compresión orientada a bit. La descripción completa de esta técnica se plantea en el capítulo siguiente. Para el caso actual se configura el modelo a través del parámetro $\alpha = 2^{10}$. Finalmente, para la compresión del texto preprocesado se consideran un conjunto de técnicas que cubre mayoritariamente las distintas estrategias de compresión posibles: las técnicas de diccionario gzip y p7zip plantean sendas propuestas basadas en variantes de los algoritmos LZ, mientras que ppmdi y bzip2 obtienen un modelo de orden superior basado, respectivamente, en PPM y la transformada de *Burrows-Wheeler*.

La tabla 4.3 muestra los resultados "en plano" de las diferentes técnicas consideradas. Estos datos representan, básicamente, el tamaño del texto preprocesado sobre la codificación orientada a byte considerada en cada una de las técnicas. WCIM representa la opción más efectiva en la mayoría de los textos analizados mostrando una ligera ganancia respecto a EDTC y E-G. Por su parte, la diferencia existente entre mPPM y el resto de técnicas es especialmente significativa dado que ésta requiere entre un 25 % y un 38 % más de espacio que las otras. Gracias a los subconjuntos de textos contemplados en las colecciones AP, WSJ y ZIFF podemos observar como la efectividad de las técnicas mejora notablemente con el tamaño del texto debido al crecimiento sublineal de su vocabulario y con independencia del corpus seleccionado (a excepción de mPPM cuya mejora es sensiblemente inferior debido a la política de reemplazo de palabras que utiliza).

A continuación, se analizan los resultados obtenidos al comprimir los textos preprocesados con cada una de las técnicas universales consideradas. Una vez conocido el comportamiento de estas técnicas con el tamaño de los textos, las pruebas actuales se llevan a cabo sobre un subconjunto más reducido de las colecciones AP, WSJ y ZIFF. Para la experimentación actual se consideran tamaños de texto de, aproximadamente, 1, 10 y 100 MB.

las ratios obtenidas al comprimir el texto directamente con gzip y p7zip. Como era esperado, de acuerdo a las propiedades con las que se construye cada uno de los algoritmos, las ratios de compresión obtenidas por p7zip son notablemente mejores a los de gzip (entre un 19 % y un 31 %). Esta gran diferencia repercute directamente en la efectividad de los resultados conseguidos al comprimir el texto preprocesado. En todos los casos, las técnicas construidas sobre compresión

Texto	gzip	p7zip	ETDC+		E-G+		mPPM+		WCIM+	
			gzip	p7zip	gzip	p7zip	gzip	p7zip	gzip	p7zip
AP001	37,63 %	28,95 %	29,48 %	26,66 %	30,60 %	28,60 %	29,84 %	**25,41 %**	28,11 %	25,76 %
AP010	37,31 %	24,91 %	26,26 %	21,78 %	26,62 %	24,45 %	28,18 %	**21,42 %**	25,53 %	21,51 %
AP100	37,34 %	24,21 %	25,31 %	**20,00 %**	24,43 %	22,08 %	27,77 %	20,07 %	24,95 %	20,06 %
WSJ001	37,07 %	29,00 %	28,93 %	26,61 %	30,13 %	28,06 %	29,48 %	**25,37 %**	27,72 %	25,77 %
WSJ010	37,13 %	26,00 %	26,30 %	22,67 %	26,50 %	24,26 %	28,25 %	**22,28 %**	25,58 %	22,41 %
WSJ100	37,24 %	25,38 %	25,30 %	**21,01 %**	24,16 %	21,95 %	27,82 %	21,09 %	24,96 %	21,05 %
ZIFF001	31,10 %	25,20 %	24,64 %	22,74 %	26,40 %	24,71 %	25,35 %	**21,88 %**	23,74 %	22,16 %
ZIFF010	33,01 %	24,60 %	24,25 %	21,45 %	25,50 %	23,56 %	26,04 %	**21,08 %**	23,61 %	21,20 %
ZIFF100	33,19 %	24,45 %	23,61 %	**20,46 %**	23,79 %	21,88 %	25,88 %	20,52 %	23,27 %	20,48 %
FT91	36,42 %	26,11 %	25,93 %	22,54 %	25,95 %	23,93 %	27,64 %	**22,13 %**	24,99 %	22,17 %
FT92	36,48 %	25,55 %	24,93 %	**21,00 %**	23,62 %	21,64 %	27,26 %	21,25 %	24,48 %	21,03 %
FT93	34,21 %	23,82 %	23,32 %	**19,49 %**	22,05 %	20,18 %	25,47 %	19,74 %	22,88 %	19,51 %
FT94	34,21 %	23,75 %	23,38 %	**19,44 %**	21,99 %	20,06 %	25,49 %	19,68 %	22,93 %	19,45 %
CR	33,29 %	22,93 %	22,38 %	18,81 %	22,16 %	20,32 %	24,11 %	**18,65 %**	21,90 %	18,75 %

Tabla 4.4: Efectividad del preprocesamiento WCIM y otras técnicas comparables sobre compresión LZ (gzip y p7zip).

p7zip obtienen unos resultados más competitivos.

mPPM+p7zip es la mejor opción para textos de menor tamaño, mejorando hasta un $1,5\,\%$ a WCIM_{p7zip} y hasta un $4{,}7\,\%$ a ETDC+p7zip. La diferencia respecto a E-G+p7zip es aún mayor dado que el preprocesamiento E-G, basado en un modelo de orden 1, no favorece su compresión con una técnica de diccionario. El número de palabras diferentes, en textos de pequeño tamaño, no completa la capacidad del diccionario utilizado en mPPM por lo que no se producen reemplazos y, con ello, no se pierde efectividad en la codificación del vocabulario. Además, la alta frecuencia de aparición del código <00000000> (como primer byte de los códigos utilizados para representar las 256 palabras de código más frecuentes) facilita identificar y manejar como frases las secuencias del tipo <00000000 xxxxxxxx> / x $\in \{0, 1\}$ y con ello representarlas de una forma más efectiva sobre los diccionarios construidos con las variantes de LZ. Sin embargo, para ficheros de mayor tamaño, la propiedad anterior se hace, progresivamente, menos significativa de tal forma que las tres técnicas tienden a alcanzar una efectividad similar, siendo ETDC+p7zip ligeramente mejor.

El resultado más destacable radica en la buena conjunción existente entre las técnicas orientadas a byte (sobre un modelo de orden 0) y p7zip. Tanto mPPM+p7zip como WCIM_{p7zip} alcanzan mejoras de hasta un $18\,\%$ respecto al resultado obtenido al aplicar directamente p7zip sobre el texto original. Esto es debido a la propia naturaleza de los compresores basados en LZ. Estas técnicas se basan en la identificación y codificación de frases en el texto. Aceptemos que una técnica basada en LZ está capacitada para identificar frases de hasta k símbolos. Esto supone que al aplicar, directamente, estas técnicas a un texto en lenguaje natural, las secuencias identificadas contendrán hasta k letras. Sin embargo, al hacerlo al resultado del preprocesamiento del texto, la secuencia contiene la información relativa de hasta $k/2$ palabras del texto original, considerando que la longitud media de una palabra de código, en las técnicas estudiadas, es de 2 (en WCIM y ETDC este valor es aún inferior). Esta propiedad es que la que sustenta la gran mejora obtenida por la combinación de preprocesamiento orientado a byte y compresión con p7zip.

La siguiente parte de este estudio experimental se centra en los resultados obtenidos con compresores basados en un modelado de orden superior. Para este caso se considera el efecto del preprocesamiento sobre compresores basados en PPM (ppmdi) y BWCA (bzip2). En términos generales, la efectividad de estas combinaciones va a ser inferior a la obtenida en el caso anterior, excepto para la técnica E-G. Esto se debe a las propiedades de modelado sobre las que se desarrollan mPPM, WCIM y ETDC. Todas ellas utilizan un modelo de orden 0 sobre una función de diccionario particular. Por su parte, E-G construye un modelo basado en grafo que permite obtener una representación de orden 1 del texto. Esto permite codificar cada palabra de acuerdo al contexto que representa su predecesora, lo cual se traduce en una distribución más sesgada

Texto	ppmdi	ETDC+ppmdi	E-G+ppmdi	mPPM+ppmdi	WCIM+ppmdi
AP001	26,43 %	26,30 %	26,46 %	**24,44 %**	24,88 %
AP010	25,59 %	22,63 %	23,26 %	22,80 %	**22,25 %**
AP100	25,64 %	21,66 %	21,65 %	22,47 %	**21,57 %**
WSJ001	25,61 %	26,03 %	25,93 %	**24,16 %**	24,70 %
WSJ010	25,42 %	22,83 %	23,02 %	22,91 %	**22,42 %**
WSJ100	25,40 %	21,82 %	**21,35 %**	22,53 %	21,70 %
ZIFF001	21,08 %	22,12 %	22,88 %	**20,67 %**	21,13 %
ZIFF010	22,97 %	21,44 %	22,40 %	21,35 %	**21,07 %**
ZIFF100	23,17 %	20,86 %	21,31 %	21,34 %	**20,76 %**
FT91	25,30 %	22,86 %	22,74 %	22,75 %	**22,28 %**
FT92	25,34 %	21,88 %	**21,16 %**	22,56 %	21,70 %
FT93	23,55 %	20,26 %	**19,64 %**	20,86 %	20,07 %
FT94	23,55 %	20,29 %	**19,55 %**	20,91 %	20,11 %
CR	22,42 %	19,71 %	19,70 %	19,97 %	**19,48 %**

Tabla 4.5: Efectividad del preprocesamiento WCIM y otras técnicas comparables sobre compresión PPM (ppmdi).

Texto	bzip2	ETDC+bzip2	E-G+bzip2	mPPM+bzip2	WCIM_{bzip2}
AP001	28,31 %	28,25 %	28,35 %	**26,37 %**	26,83 %
AP010	27,20 %	23,92 %	24,87 %	23,91 %	**23,56 %**
AP100	27,23 %	22,87 %	23,18 %	23,60 %	**22,82 %**
WSJ001	27,45 %	27,78 %	27,76 %	**25,97 %**	26,48 %
WSJ010	26,92 %	24,09 %	24,59 %	24,08 %	**23,76 %**
WSJ100	26,86 %	23,00 %	**22,90 %**	23,74 %	22,99 %
ZIFF001	23,29 %	23,69 %	24,77 %	**22,28 %**	22,83 %
ZIFF010	25,00 %	22,85 %	24,18 %	22,76 %	**22,58 %**
ZIFF100	25,21 %	22,26 %	23,03 %	22,74 %	**22,23 %**
FT91	27,06 %	24,22 %	24,26 %	24,11 %	**23,65 %**
FT92	27,10 %	23,15 %	**22,66 %**	23,76 %	22,97 %
FT93	25,32 %	21,51 %	**21,10 %**	22,17 %	21,41 %
FT94	25,27 %	21,44 %	**21,01 %**	22,18 %	21,39 %
CR	24,14 %	20,93 %	21,14 %	21,24 %	**20,75 %**

Tabla 4.6: Efectividad del preprocesamiento WCIM y otras técnicas comparables sobre compresión BWCA (bzip2).

de los primeros bytes en el alfabeto de salida.

En primer lugar se muestran los resultados obtenidos con ppmdi (tabla 4.5) y se analiza el hecho anterior sobre las propiedades del modelo de predictivo en el que se basa PPM. Se puede ver como mPPM+ppmdi sigue siendo la mejor opción para los textos más pequeños. mPPM+ppmdi mejora hasta un 2 % a WCIM_{ppmdi}, un 7 % a ETDC+ppmdi y hasta cerca de un 10 % a EG+ppmdi, cuya efectividad está lastrada por la codificación del grafo que utiliza como modelo del texto.

Sin embargo, para el siguiente tamaño de texto (10MB) WCIM_{ppmdi} ya se consolida como la mejor opción, excepto para los textos de mayor tamaño (FT92, FT93 y FT94 y WSJ100). Para los textos de 10MB, mPPM+ppmdi paga el precio de utilizar un gran porcentaje de la capacidad de su diccionario. Mientras tanto, ETDC+ppmdi presenta una efectividad inferior a la de WCIM_{ppmdi} aunque ambas se aproximan al aumentar el tamaño del texto dado que la ganancia debida a las heurísticas planteadas por WCIM disminuye con el crecimiento del diccionario. Finalmente, E-G está sujeta, en todo momento, a la necesidad de codificar su grafo de representación. Esto supone un coste importante para tamaños pequeños. Aún así, con el crecimiento del texto, las mejores predicciones obtenidas por ppmdi llegan a compensar el coste de codificar el grafo de tal forma que se convierte en la mejor opción para los ficheros de gran tamaño (obtiene una mejora de hasta un 2, 84 % respecto a WCIM_{ppmdi} para el texto FT94). En términos globales, WCIM_{ppmdi} mejora la efectividad de ppmdi entre un 3 % y un 16 %.

Finalmente, la tabla 4.6 muestra los resultados obtenidos al utilizar bzip2 como compresor del texto preprocesado. La interpretación de dichos resultados es similar a la planteada para

ppmdi. Para este caso, la mejora de $\texttt{WCIM}_{bzip2}$ respecto a bzip2 se sitúa entre un 2 % y un 16 % lo que supone un comportamiento muy cercano al obtenido para el caso de ppmdi.

4.5 Conclusiones y trabajo futuro

Las técnicas que conjugan las propiedades de un preprocesamiento del texto con su posterior compresión sobre una propuesta universal, han ganado importancia respecto al diseño de técnicas *ad hoc.* para compresión de texto. La flexibilidad de este tipo de técnicas permite un manejo eficaz de su *trade-off* espacio/tiempo que puede ser adaptado (sobre el compresor universal considerado a la salida) a las necesidades específicas del contexto de aplicación. La gran mayoría de estas técnicas utilizan una representación del texto basada en diccionario y orientada a palabras. Sobre este modelo, sugieren diferentes heurísticas para la detección y codificación efectiva de las regularidades identificadas en el texto. Sin embargo, estas ideas se centran, en general, en aspectos de bajo nivel que posibilitan pequeñas mejoras en la efectividad de las técnicas resultantes.

La técnica WCIM, propuesta en el capítulo actual, adopta un enfoque diferente al anterior basándose en la experiencia previa adquirida con mPPM. Esta técnica se basa en la utilización de un diccionario de tamaño limitado y en la codificación del texto preprocesado sobre un esquema orientado a byte. Por su parte, WCIM se centra en la organización de un diccionario cuyo tamaño se limita de acuerdo a la combinación del conocimiento derivado de la ley de Heaps y a un nuevo esquema de codificación orientado a byte. Para ello desarrolla cuatro heurísticas complementarias basadas en regularidades detectadas en los vocabularios que forman un texto en lenguaje natural y en la distribución de frecuencia que presentan. Estas heurísticas favorecen la codificación de las palabras más frecuentes y ofrecen un tratamiento específico para las restantes, manteniendo fuera del diccionario las palabras con una única ocurrencia.

La experimentación llevada a cabo ha mostrado propiedades interesantes tanto en WCIM como en mPPM, cuya efectividad había sido probada, únicamente, con compresión basada en PPM. Ambas técnicas obtienen una representación muy compresible con algoritmos de la familia LZ, más concretamente con lzma (implementado en el compresor p7zip). En este caso, las mejoras alcanzan hasta un 18 % respecto al propio p7zip, mientras que el beneficio obtenido con con compresores basados en modelos de orden superior alcanzan el 16 % para ppmdi y bzip2. En términos globales, el preprocesamiento obtenido por WCIM representa una solución competitiva para diferentes tipos de compresores cuya efectividad se mejora de forma notable. $\texttt{WCIM}_{p7zip}$ representa la opción más competitiva para textos de gran tamaño, dado que p7zip es capaz de obtener un mejor aprovechamiento de la transformación, orientada a byte, que representa el texto preprocesado. Esta propiedad plantea algunas ideas interesantes para el trabajo futuro.

Por un lado, se considera la integración de WCIM y las propiedades de mPPM deducidas en la experimentación actual. Por un lado, la utilización de una lista de espera permite evitar la recodificación de palabras desalojadas en un diccionario similar al de mPPM cuya ocupación se puede reducir con el tratamiento específico de palabras "unitarias". Esta decisión transforma mPPM en una técnica semi-estática cuya efectividad mejorará, previsiblemente, la conseguida con $\texttt{WCIM}_{p7zip}$. Complementariamente, se considera la posibilidad de añadir nuevas heurísticas basadas en la extensión del alfabeto de entrada con q-gramas significativos. Este tipo de propuestas persiguen reducir el número total de símbolos representados a costa de un un crecimiento limitado del vocabulario. Estas heurísticas pueden ser incorporadas tanto en mPPM como en WCIM. Esto supone mejorar la efectividad actualmente alcanzada por las técnicas $\texttt{WCIM}_{ppmdi}$ y $\texttt{WCIM}_{bzip2}$.

Finalmente, se plantea la posibilidad de utilizar estas heurísticas en otros contextos relacionados con la compresión. La heurística centrada en la gestión de palabras "unitarias" puede tener una mayor utilidad en distribuciones de símbolos de mayor tamaño que las actuales. En

la experimentación planteada se observa que las palabras "unitarias" representan entre un 33 % y 59 % del tamaño total del vocabulario. Una gestión independiente de estas palabras permite aliviar la carga de símbolos en el diccionario y, a su vez, reducir la longitud media de palabra de código utilizada. En contextos, como la representación de `RDF`[8] el tamaño del diccionario utilizado para la representación de los símbolos identificados en los triples tiende a ser mucho mayor que el necesario en lenguaje natural. Por ejemplo, un *chunk* del corpus `Billion`[9] formado por 10 millones de triples, contiene 30 millones de símbolos de los cuales un $10,45\,\%$ representa su vocabulario. Una primera aproximación a la compresión de este contenido reduce hasta un 40 % el tamaño del diccionario finalmente obtenido al considerar la heurística actual para la representación alternativa de aquellos símbolos que aparecen una única vez en el grafo `RDF` tratado, con la consiguiente mejora, en efectividad, que esto supone.

[8] http://www.w3.org/RDF/
[9] http://challenge.semanticweb.org/

El peligro
es el fantasma que planea
sobre aquello
que juraste un día alcanzar.
Carlos Goñi

5

Modelado y Codificación basada en Aristas (Edge-Guided)

Hace unos veinte años, Witten y Bell [WB90] describieron los modelos del lenguaje como herramientas "fascinantes" por su propia naturaleza y con independencia de su contexto de aplicación. El paso de los años ha demostrado lo acertado de sus expectativas. El desarrollo y la evolución de estos modelos del lenguaje ha traído consigo la aparición y asentamiento de nuevas disciplinas en el campo de la informática. En el citado trabajo, los autores señalan la compresión de texto, la identificación de autoría o los programas de corrección ortográfica como algunas de sus aplicaciones. Cada una de estas áreas se ha asentado sobre diferentes propuestas de modelado que, a su vez, han ayudado al desarrollo de una disciplina completa dedicada al Procesamiento del Lenguaje Natural (PLN). Aplicaciones de extracción de información, categorización de contenidos o traducción automática, entre otras, se han desarrollado en el ámbito del PLN, sobre diferentes modelos del lenguaje cuyo diseño enfoca las necesidades específicas de cada área de investigación.

Witten y Bell definen, en el citado trabajo, los modelos del lenguaje como *colecciones de información que aproximan las estadísticas y la estructura del texto* que está siendo modelado. Además consideran que, a pesar de la increíble complejidad del lenguaje, la utilización de diferentes técnicas de modelado habilita la identificación de cantidades significativas de estructura aunque advierten acerca de que la regularidades detectadas desde una perspectiva estadística no son completamente fiables.

Chen [Che96] define un *modelo del lenguaje* como una "descripción del propio lenguaje" que puede ser formalizada, desde una perspectiva lingüística, mediante gramáticas, o a través de un modelo probabilístico, siguiendo la propuesta de Witten y Bell. A la hora de diseñar un modelo del lenguaje se debe considerar el entorno particular en el que éste va a ser utilizado. Esto facilita la obtención de una descripción precisa de aquellas propiedades del lenguaje demandadas por la aplicación y, a su vez, permite descartar otras de menor importancia, relajando con ello la complejidad del modelo final.

Un modelo del lenguaje debe ser comparado respecto a otros modelos similares construidos para su uso en un entorno similar. Esta comparación debe ser capaz de evaluar no sólo la efectividad del modelo, sino también su eficiencia, considerando ambos criterios dentro del entorno de aplicación en el que se opera.

Existen numerosos trabajos centrados en los modelos de lenguaje por ordenador, la mayoría de ellos aplicados al idioma inglés. También son múltiples las aplicaciones en las que éstos han sido utilizados. En la siguiente sección se plantea una breve revisión de algunos modelos de orden k utilizados para compresión de texto. Aún así, cabe destacar como diversos modelos del lenguaje, diseñados originalmente para compresión de texto, han sido adaptados para su utilización en otras áreas. La tesis de W.J. Teahan [Tea97] es una rica fuente de información al respecto. En ella se enfoca el desarrollo de modelos del lenguaje para el idioma inglés utilizando diferentes variantes de `PPM` tanto para la compresión de texto como para otras aplicaciones

relacionadas con la criptología, la corrección ortográfica, el reconocimiento de voz o la corrección del propio idioma [TICH98], considerando que PPM es capaz de predecir texto en inglés casi tan bien como un humano [TC96]. Más recientemente, los modelos del lenguaje basados en PPM también se han mostrado como soluciones eficaces para la clasificación de texto basada en contenido [Bob06], considerando que los mejores descriptores, para un determinado texto, no tienen porqué ser palabras únicas sino que pueden estar formados por la combinación de diferentes palabras.

El trabajo presentado en este capítulo toma, como punto de partida, la idea que sugiere el diseño de una técnica de compresión que combine la alta velocidad de una codificación basada en diccionario con la excelente compresión que puede obtenerse mediante un esquema probabilístico de contexto finito [BCW90, 151]. En 1994, Gutmann y Bell [GB94] plantean, sobre la idea anterior, el diseño de una técnica de compresión *híbrida* que utiliza una técnica de diccionario sobre un modelo contextual. Los autores concluyen la cercanía existente entre los métodos estadísticos y de diccionario y sugieren ciertas alternativas como la posibilidad de predecir un conjunto de símbolos a partir de un contexto o la utilización de códigos de longitud variable para favorecer la representación de frases recientemente procesadas. Ambas ideas se consideran como base para el desarrollo de la presente propuesta.

El capítulo actual plantea cómo el texto se puede modelar y codificar a través de la información almacenada en las aristas de un grafo (*Edge-Guided*) utilizado como modelo para su representación. La utilización de un grafo dirigido de palabras (para el caso actual se considera la caracterización de palabra de *palabra* dada en la Definición 3.1) permite almacenar una descripción estadística del texto en las aristas de la estructura. Esto supone que cada palabra identificada en el texto se representa y codifica utilizando como contexto la palabra justo anterior, de tal forma que el grafo almacena una caracterización estadística del texto basada en los pares de palabras adyacentes que lo constituyen. El desarrollo de la propuesta actual se compone de dos etapas. La primera de ellas establece el diseño de una técnica de orden 1 como caso base para la propuesta global. En este caso inicial se plantea una propuesta eficiente para la implementación del modelo del lenguaje que caracterizamos a nivel conceptual. El presente desarrollo se lleva a cabo sobre un modelado adaptativo en el que se integra una propuesta de codificación capaz de aprovechar de forma efectiva la información representada en el modelo. Esta experiencia previa se utiliza como base para la ejecución de la segunda etapa en la que se analiza la extensión del modelo original de forma que soporte la representación de estadísticas de orden superior. Este trabajo se fundamenta en la idea de construir un modelo de orden 1 orientado a frases sobre una extensión del alfabeto de entrada cuyos símbolos pasan a ser secuencias de una o más palabras. Esto supone que el modelo original de orden 1 pasa a un orden k desde una perspectiva orientada a palabras. La transición de uno a otro modelo se fundamenta en la utilización de una gramática libre de contexto que permite representar la jerarquía de palabras subyacente a las frases consideradas para la extensión del vocabulario. El resultado de esta segunda etapa extiende la técnica original a través de su esquema de codificación. La nueva propuesta mantiene las propiedades del esquema original y añade una función de codificación centrada en la representación de la jerarquía inherente a la gramática.

El presente capítulo plantea, inicialmente, una revisión del trabajo relacionado con las ideas que caracterizan las propuestas actuales (§5.1) y un breve resumen sobre Teoría de Grafos (§5.2) en el que se definen todos aquellos conceptos utilizados a lo largo de este capítulo. Las dos secciones siguientes organizan cada una de nuestras propuestas a partir de las propiedades que caracterizan su modelado, codificación y detalles técnicos de implementación. La Sección §5.3 presenta el caso base que da lugar a la obtención de un conjunto de compresores de orden 1. Esta experiencia previa se utiliza para la generalización del modelo original, en la Sección §5.4, obteniendo una representación de orden superior sobre el texto procesado. Un análisis

experimental detallado sobre las propuestas anteriores (§5.5) y las conclusiones extraídas de esta parte del trabajo de tesis (§5.6) completan el capítulo actual.

5.1 Trabajo relacionado

Los modelos de orden k obtienen una representación estadística del texto basada en la probabilidad de aparición de un símbolo en el contexto establecido por los k símbolos precedentes. Cómo se planteaba en §3.4.1, `PPM` [CW84b] es una técnica de compresión desarrollada sobre un modelo de orden k orientado a caracteres. Aunque estos modelos orientados a caracteres no son, en general, capaces de aprovechar la relación de adyacencia existente entre dos palabras consecutivas, se ha demostrado que `PPM` es capaz de predecir texto en inglés casi tan bien como un humano [TC96]. Esto se traduce en unas excelentes ratios de compresión cuando las técnicas basadas en `PPM` se utilizan sobre texto en lenguaje natural.

La construcción de un modelo `PPM` de un orden superior (7, 8 ó 9 letras) se puede afrontar de manera práctica en recursos de tiempo y memoria, dado que los contextos que los forman presentan un número limitado de relaciones con otros contextos. Sin embargo, al intentar construir un modelo de orden superior orientado a palabras se plantea un importante problema de recursos considerando el gran número de contextos que deben ser mantenidos al utilizar un alfabeto de tamaño mayor [Mof89]. Esto supone que el diseño y, sobre todo, la implementación de modelos del lenguaje basados en estadísticas de orden superior se convierta en un problema complejo. Sin embargo, éste es también un problema interesante dentro de aquellos entornos en los que la efectividad de la compresión es el objetivo principal.

La sección actual enfoca, en primer lugar, el análisis de diferentes técnicas de orden superior orientadas a palabras. Complementariamente se considera una segunda línea de investigación, afín al contexto actual, que recoge algunas de las técnicas basadas en la transformación del texto y su posterior compresión con una técnica universal, estudiadas en el capítulo anterior.

Horspool y Cormack [HC92] sugieren el uso de un modelo de orden 1 orientado a palabras, siguiendo la propuesta original de Bentley *et al.* [BSTW86]. Esto supone el diseño de un modelado basado en *alfabetos separados* de tal forma que que la representación de cada palabra y cada separador se lleva a cabo, respectivamente, de acuerdo a la palabra y al separador precedentes. La correlación existente entre lo símbolos es diferente en ambos casos, siendo más fuerte entre las palabras que entre los separadores. El modelo se implementa sobre un mecanismo de *blending* similar al utilizado en `PPMC` [BCW90] que permite combinar las estadísticas de los modelos de orden 0 y orden 1. Finalmente, los autores categorizan las palabras en *parts-of-the-speech* mejorando con ello las estadísticas representadas en el modelo original. Teahan y Cleary proponen sendos modelos de orden 1 basados en palabras y *parts-of-the-speech* [TC97a]. Ambas propuestas utilizan el mecanismo de *blending* de `PPM` sobre una variante del método X (denominada $X1$) que estima la probabilidad del símbolo de escape de forma proporcional al número de palabras que han aparecido sólo una vez en el contexto. El rendimiento de estos modelos mejora sobre dos mecanismos denominados *full exclusions* (en el que se excluyen las palabras predichas en un contexto superior) y *update exclusions* (que sólo actualiza los contadores en los contextos sobre los que realmente se hace la predicción). Ambas propuestas mejoran, hasta un 3-4 %, los resultados obtenidos por un compresor `PPM` de orden 5.

Moffat [Mof89] extiende el modelo original de `PPM` para soportar la representación de palabras sobre la referida propuesta de Bentley *et al.* La experimentación llevada a cabo con esta nueva propuesta demuestra que la efectividad de la técnica mejora notablemente para el modelo de primer orden mientras que para el de orden 2 la mejora es más limitada. En esta misma línea encontramos un trabajo que extiende el modelo `PPM` para el manejo de palabras como símbolos de su alfabeto fuente [ÁCPFL09]. Esta técnica, denominada `SWPPM`, tiene una naturaleza semi-

estática que le permite eliminar el coste de adaptación inicial que requiere un modelo PPM y que puede ser demasiado elevado para una representación basada en palabras. Además, establece un proceso de filtrado que permite ahorrar la representación de aquellos contextos menos significativos respecto a un umbral determinado. Finalmente, SWPPM utiliza un modelo recursivo para el almacenamiento de contextos de forma que obtiene, para cada uno de ellos, una jerarquía en la que el contexto de mayor orden actúa como raíz del árbol derivación. A pesar de lo interesante de las ideas comentadas, los resultados obtenidos (entre el 28 % y 31 %) no llegan a mejorar la efectividad de un PPM de caracteres.

En esta misma línea queremos destacar un último trabajo [BW99a, BW99b] que desarrolla un modelo de orden k sobre el concepto de *atracción léxica* [Yur98]. Se entiende por atracción léxica de dos palabras la probabilidad de que éstas aparezcan dentro de una misma frase con independencia de su orden y de la distancia existente entre ellas. Los autores afirman, de acuerdo a la propiedad anterior, que los mejores predictores para una palabra no tienen porqué ser sus predecesores inmediatos y sugieren la utilización de una medición basada en atracción léxica para la identificación de estructuras de dependencia que permitan obtener una representación del texto con una baja entropía. Sin embargo, los resultados obtenidos por esta técnica no son capaces de mejorar, en algunos casos, los obtenidos al utilizar modelos de orden 0, dada la importante sobrecarga que supone codificar el grafo de dependencia utilizado para representar el texto. Con el fin de mejorar la efectividad de la técnica, los autores consideran la penalización de las relaciones existentes entre palabras no adyacentes y sugieren la posibilidad de limitar el número de relaciones de una palabra. Esto permite una importante reducción en el número de bits necesarios para la codificación del modelo a costa de perder calidad en la información mutua que éste representa. Esta transformación da lugar a un modelo en el que se refleja el orden natural de las palabras y la información mutua representada es equivalente a la obtenida por un modelo PPM de orden 1 orientado a palabras.

Dado el alcance de la propuesta actual, es importante reseñar la existencia de otras técnicas de compresión que extienden con secuencias de palabras el alfabeto fuente original. El éxito de este tipo de técnicas radica en su habilidad a la hora de seleccionar "buenas" frases que permitan obtener una mejora de la información representada en el modelo. Como se comentaba en la Sección §3.5.2, la selección del conjunto de frases que optimiza la compresión de un texto es un problema $\mathcal{NP}$-difícil [SS82] que precisa del uso de heurísticas para aproximar la obtención de un subconjunto efectivo de estas frases. Los compresores basados en gramáticas libres de contexto plantean una alternativa competitiva para este problema como demuestra la adaptación del algoritmo Re-Pair presentada por Raymond Wan [Wan03] para su uso con textos en lenguaje natural. Una nueva técnica [BFL$^+$10] (*variable-to-variable compressor*: v2vdc), de muy reciente publicación, afronta la detección de las frases mediante la construcción del *array de sufijos* del texto y el uso de la estructura *Longest Common Preffix* ($\mathcal{LCP}$) para la elección de las frases relevantes. Esta selección atiende a criterios de frecuencia y longitud del prefijo. La técnica v2vdc obtiene una efectividad próxima a la conseguida por compresores como ppmdi, p7zip o Re-Pair, permitiendo, a su vez, la búsqueda sobre el texto comprimido mediante un esquema específico basado en el algoritmo Set-Horspool (ver §3.2.2) capaz de aprovechar las propiedades del código denso ETDC (ver §3.3.3) utilizado para la codificación del texto sobre el alfabeto de entrada extendido.

Paralelamente a este conjunto de propuestas se consideran las comentadas en el capítulo anterior. Estas técnicas transforman el texto original en una secuencia de bytes altamente compresible mediante una técnica universal orientada a bit. Sus propiedades permiten la identificación de correlaciones entre palabras consecutivas con un valor menor de k que las soluciones universales o capturan correlaciones de mayor tamaño para un valor de k determinado. Este hecho permite mejorar, significativamente, la efectividad de las técnicas universales consideradas

para la compresión del texto preprocesado obteniendo aproximaciones a una compresión de orden superior orientada a palabras. Las técnicas incluidas en el contexto actual son las analizadas en el capítulo anterior.

5.2 Conceptos básicos

La sección actual plantea una breve revisión de los conceptos básicos de *Teoría de Grafos* [Die05] necesarios para la definición de la técnica `Edge-Guided` actual.

Definición 5.1 (Grafo) *Un grafo $\mathcal{G}$ está formado por un par de conjuntos finitos $\mathcal{G}(\mathcal{N}, \mathcal{A})$, donde $\mathcal{N}$ es el conjunto de nodos (vértices) y $\mathcal{A}$ es el conjunto de aristas definidas como pares (n, n') / $n, n' \in \mathcal{N}$.*

Esto supone que un grafo describe un conjunto de conexiones entre objetos. Cada uno de estos objetos es un *nodo* o *vértice*, mientras que las conexiones existentes entre ellos se denominan *aristas*.

Definición 5.2 (Par ordenado) *Un par ordenado describe una colección de dos objetos en la que éstos pueden ser distinguidos como primer y segundo elemento del par. Un par formado por x como primer elemento e y como segundo se representa como (x, y).*

Definición 5.3 (Grafo dirigido) *Se supone un conjunto de nodos $\mathcal{N}$ y un conjunto de pares ordenados $\mathcal{A}$, esto es un subconjunto de $\mathcal{N} \times \mathcal{N}$, denominados arcos o aristas dirigidas. Un grafo dirigido (o digrafo) se define como un par ordenado $\mathcal{G} := (\mathcal{N}, \mathcal{A})$, donde $\mathcal{N}$ es el conjunto que contiene todos los nodos que forman parte de $\mathcal{G}$ y $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ el conjunto de todas sus aristas.*

Por tanto, el *orden* es una propiedad significativa en un par ordenado, lo cual supone que un par de objetos (x, y) es diferente de (y, x), para $x \neq y$. En Teoría de Grafos, el primer nodo del arco se denomina *origen* mientras que el segundo se refiere como *destino*. Finalmente, es importante indicar que un grafo dirigido puede contener diferentes arcos entre dos mismos nodos x e y, denominados *arcos múltiples*.

Definición 5.4 (Conjunto de Entrada) *Se supone un grafo dirigido $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ en el que $n, n' \in \mathcal{N}$. El conjunto de entrada para un nodo $n \in \mathcal{N}$ contiene todos aquellos arcos cuyo destino es n y se define como $\mathcal{D}^+(n) := \{a \in \mathcal{A} / \exists n' \in \mathcal{N} : a = (n', n)\}$.*

Definición 5.5 (Conjunto de Salida) *Se supone un grafo dirigido $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ en el que $n, n' \in \mathcal{N}$. El conjunto de entrada para un nodo $n \in \mathcal{N}$ contiene todos aquellos arcos cuyo origen es n y se define como $\mathcal{D}^-(n) := \{a \in \mathcal{A} / \exists n' \in \mathcal{N} : a = (n, n')\}$.*

Definición 5.6 (Grado) *El grado de un nodo se define como el número de aristas conectadas a él en el grafo. Para el caso de grafos dirigidos, el grado de entrada de $n \in \mathcal{N}$ se define como el número de arcos en los que n actúa como destino, mientras que el grado de salida contempla aquellos arcos en los que n actúa como fuente. De esta forma, el grado de un nodo en un grafo dirigido se obtiene como la suma de sus grados de entrada y salida.*

Por tanto, los grados de entrada y salida de $n \in \mathcal{N}$ se definen, respectivamente, como $deg^+(n) = |\mathcal{D}^+(n)|$ y $deg^-(n) = |\mathcal{D}^-(n)|$.

Definición 5.7 (Camino) *Se define un camino, en un grafo dirigido, como una secuencia $n_0, a_1, n_1, \ldots, a_n, n_n$ que alterna nodos y arcos, en la que cada arco a_i es (n_{i-1}, n_i).*

En el caso actual, n_0 y n_n se denominan, respectivamente, *nodo inicial* y *nodo final* del camino. Un camino puede representarse de forma concisa mediante las secuencias de arcos ($< a_1, a_2, \ldots, a_n >$) o nodos que lo definen ($< n_0, n_1, \ldots, n_n >$). La *longitud* de un camino se define como el número arcos que se recorren en él (n). Representaremos como $\mathcal{W}(G)$ el conjunto de todos los caminos en G. Si existe un camino $w = x, n_1, \ldots, n_j, y$, con x e y como nodos inicial y final, respectivamente, se representa como $x \overset{w}{\rightsquigarrow} y$. Esta representación se cambia por $x \rightsquigarrow y$ para aquellos casos en los que w no es relevante.

5.3 Propuesta de orden 1 orientada a palabras

Los textos en lenguaje natural contienen información utilizada para la comunicación humana de propósito general (ver §3.1). Adoptar una estrategia *antropomórfica*, sugerida para el desarrollo de modelos del lenguaje por ordenador [KdM90], facilita afrontar la identificación de regularidades en un texto. Esta estrategia se basa en entender el comportamiento humano ante una determinada actividad y, posteriormente, incorporar este conocimiento al modelo.

La premisa principal, ante un acto genérico de comunicación humana, sugiere que las personas no interpretan un mensaje como una secuencia de caracteres aislados sino como los conjuntos de caracteres alfanuméricos que forman las *palabras*. Por tanto, la comunicación humana utiliza la palabra como unidad mínima de información lo que supone que los modelos del lenguaje orientados a caracteres no sean capaces, en la mayoría de los casos, de detectar y aprovechar la relación existente entre dos palabras consecutivas. Considerando que la compresión de texto trata de identificar las regularidades existentes en los textos procesados, parece interesante la idea de plantear un modelo del lenguaje basado en la información derivada de la relación existente entre las palabras que conforman el mensaje. Un sencillo ejemplo soporta la afirmación anterior. Supongamos un alfabeto formado por tres palabras: "*Enrique*", "*estaba*" y "*allí*". Frases cómo "... estaba allí Enrique ..." o "... Enrique estaba allí ..." están compuestas por las mismas palabras y sin embargo no transmiten la misma información. La primera de ellas forma parte de una oración interrogativa mientras que la segunda corresponde a una oración enunciativa afirmativa. Un modelo del lenguaje que no considere las relaciones entre las palabras (por ejemplo el modelo de orden 0 sobre el que se desarrolla la propuesta presentada en el capítulo anterior) codificaría de forma similar las oraciones anteriores cuando, evidentemente, no transmiten la misma información.

Ante esta situación, el uso de una estructura de *grafo* sugiere un planteamiento natural para la representación de un texto a partir de las relaciones existentes entre sus unidades constituyentes, seleccionadas éstas de acuerdo a diferentes niveles de granularidad [NS06]. Las propiedades con las que se definen estas unidades dependen fuertemente de la aplicación que se va a desarrollar, aunque, para el caso actual, la utilización de *palabras* cómo símbolos del alfabeto de entrada representa la elección natural dada la premisa anteriormente considerada. Asimismo, es la aplicación la que determina el tipo de relaciones establecidas entre los nodos y, por ende, la naturaleza de la información almacenada en el modelo.

Tomando este contexto, como punto de partida, la Sección §5.3.1 especifica detalladamente cómo se diseña nuestro modelo basado en grafo para la representación del texto a través de la información contenida en las relaciones de adyacencia existentes entre las palabras. A partir de esta caracterización, la Sección §5.3.2 describe un esquema de codificación completo basado en la información representada en el modelo. La Sección §5.3.3 detalla las estrategias seguidas para la gestión eficiente de las aristas, indicando cómo éstas se integran en el modelo propuesto y cómo favorecen la efectividad del esquema de codificación sugerido. Finalmente, en §5.3.4 se especifica cómo se ha llevado a cabo la implementación del modelo tanto en lo que respecta a las estructuras de datos y algoritmos, cómo al esquema de codificación. Para éste último se consideran sendas

propuestas para el alfabeto de salida (§5.3.5). Como resultado de esta sección se plantean dos prototipos funcionales que implementan sendos compresores, de naturaleza adaptativa, sobre las ideas desarrolladas.

5.3.1. Modelado

La fase de modelado se centra en obtener una representación del texto sobre una determinada distribución de probabilidad (*modelo*) que, posteriormente, se utiliza en la fase de codificación. La primera decisión, a tener en cuenta a la hora de diseñar un modelo del lenguaje se centra en caracterizar el *alfabeto de entrada* (Σ) y, por tanto, en definir qué se considera como un *símbolo* en el modelo. Teniendo en cuenta que el objetivo actual es la compresión de texto en lenguaje natural, la utilización de un alfabeto de palabras es la elección natural.

Eisner y Smith [ES05] demuestran, experimentalmente, que todas las palabras que forman una frase están relacionadas entre sí dentro de un determinado tamaño de ventana, fuera del cual esta relación es poco significativa o nula. Esta propiedad justifica la utilización de representaciones basadas en grafos fuertemente conectados, en los que las relaciones significativas entre las palabras se representan en las aristas. Este tipo de modelos se han utilizado previamente, para compresión de texto, en técnicas como la basada en *atracción léxica*. Analizando sus resultados se puede observar cómo la identificación de relaciones "extendidas" entre las palabras de una frase requiere un modelado semi-estático. Esto, a su vez, conlleva la necesidad de codificar el grafo de relaciones utilizado como modelo con el objetivo de sincronizar los procesos de compresión y descompresión. Esta necesidad supone una importante sobrecarga en el tamaño del texto comprimido, lo que reduce la efectividad de la técnica como ha quedado probado en el caso del compresor basado en *atracción léxica*.

Estos mismos resultados [ES05] demuestran que las palabras más dependientes en una frase tienden a estar cercanas en ella. Esto es, si la j-ésima palabra de una frase depende de la i-ésima, entonces |i-j| es un valor cercano a 1, demostrando que la relación de adyacencia tiende a ser la más fuerte. Este hecho complementa los resultados obtenidos por la técnica basada en *atracción léxica*, de los cuales puede deducirse que la pérdida en información mutua derivada de no utilizar conexiones entre palabras no adyacentes se compensa con el ahorro conseguido en la codificación del grafo utilizado como modelo. Esta propiedades guían el diseño del presente modelo de representación. Al igual que en el capítulo anterior, la propuesta actual procesa el texto de acuerdo a la transformación *spaceless words* [dMNZ97].

Nuestro modelo del lenguaje se conceptualiza a través de una metáfora de "*red de palabras*" basada en la utilización de un modelo único que representa el texto de acuerdo al orden natural que siguen las palabras que lo conforman. Esta metáfora se materializa mediante las relaciones de adyacencia existentes entre las palabras que forman una frase, añadiendo las relaciones que vinculan la última palabra de cada frase con la primera palabra de la siguiente, de tal forma que todo el texto quede conectado en una única representación. Elegimos una estructura de *grafo dirigido* $\mathcal{G} := (\mathcal{N}, \mathcal{A})$ como base para la construcción del modelo que se lleva a cabo de acuerdo a las siguientes propiedades:

- Cada palabra w_x se representa en un nodo $x \in \mathcal{N}$ identificado por un valor entero.
- Cada transición entre dos palabras consecutivas w_x y w_y (respectivamente representadas en los nodos $x, y \in \mathcal{N}$) se modela mediante una arista dirigida (x, y), localmente identificada en el nodo x (origen) por un valor entero i.
- No se permiten arcos múltiples en G. Esto supone que únicamente se utiliza un arco $(x, y) \in \mathcal{A}$ para la representación de todas las ocurrencias en las que w_x y w_y (respectivamente representadas en los nodos $x, y \in \mathcal{N}$) sean adyacentes en el texto.

La información estadística del texto se representa en las aristas dirigidas ("aristas" de aquí en adelante) que forman parte del grafo obtenido de acuerdo a las propiedades anteriores. Esta información se calcula utilizando como contexto el nodo "origen" de la arista, de tal forma que dadas dos palabras w_x y w_y (respectivamente representadas en los nodos $x, y \in \mathcal{N}$), la arista $(x, y) \in \mathcal{A}$ almacena el número de veces que la transición de w_x a w_y se identifica en el texto. Esto supone que $\forall x \in \mathcal{N}$, el nodo posee una colección de aristas $\nu(x)$ (denominada *vocabulario de transiciones*), tal que $\nu(x) = \{k \ / \ (x, k) \in \mathcal{A}\}$, contiene todas las transiciones de w_x a w_k identificadas en el texto. Esto supone que el almacenamiento de $\mathcal{A}$ se distribuye en los vocabularios ν asociados a cada nodo en el grafo.

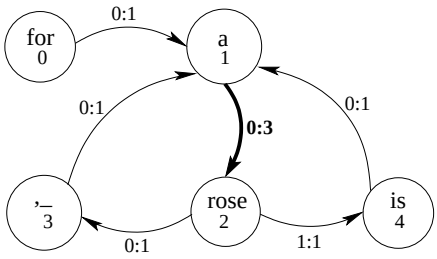


Figura 5.1: Modelado en Edge-Guided de "*for a rose, a rose is a rose*".

La figura 5.1 muestra el modelo obtenido al representar la frase "*for a rose, a rose is a rose*" de acuerdo a las propiedades anteriores. Cómo puede observarse, cada palabra está almacenada en un nodo identificado por un valor entero ("*for*" en el nodo 0, "*a*" en el 1, "*rose*" en el 2...) y las relaciones de adyacencia encontradas en el texto se representan en las aristas. Cada una de las aristas almacena dos valores en la forma $a : b$. En este caso, a contiene el identificador de la arista en el vocabulario de transiciones del nodo origen y b almacena el número de veces que la arista ha sido recorrida en el texto. Por ejemplo, el nodo 2 posee dos aristas en su conjunto de salida: $(2, 3)$ (representando la transición de "*rose*" a "*,_*") está identificada por el valor 0 mientras que la arista $(2, 4)$ (representando la transición de "*rose*" a "*is*") se identifica por el valor 1. En ambos casos, las dos transiciones han sido recorridas una sóla vez.

La información almacenada en los diferentes $\nu(x) \ / \ x \in \mathcal{N}$ proporciona una representación apropiada para el diseño de técnicas de compresión sobre el modelo actual. Sin embargo, esta representación debe ser gestionada eficientemente de tal forma que las aristas representadas en el modelo se puedan identificar de forma efectiva en sus respectivos nodos origen.

5.3.2. Codificación

El esquema de codificación propuesto se construye de forma directa sobre la información almacenada en el grafo. El resultado de la codificación actual debe contener tanto la representación comprimida del texto como la información necesaria para que el descompresor sea capaz de reconstruir el modelo utilizado. Esto da lugar a la necesidad de definir sendos esquemas para la codificación del modelo y del texto.

Codificación del modelo

La construcción del modelo se lleva a cabo sobre tres propiedades básicas: 1) cada palabra se representa en un nodo; 2) cada transición entre dos palabras consecutivas en el texto se representa en un arista que une los nodos en los que se almacena cada una de las palabras; y 3) no se permiten aristas múltiples en el modelo. Sin embargo, la tercera de ellas se establece cómo una restricción del proceso. Esto supone que el descompresor sólo necesita conocer cuándo

añadir un nuevo nodo o una nueva arista al modelo, dado que las estadísticas asociadas a las aristas se recalculan fácilmente durante su reconstrucción.

Considerando que la información estadística almacenada en el modelo se calcula utilizando como contexto el nodo "origen" de cada arista, la representación de un nuevo nodo o una nueva arista debe considerar como contexto el nodo en el que se almacena la palabra procesada en el paso anterior. Por lo tanto, ambas situaciones están sujetas a las mismas propiedades que definen el *problema de la frecuencia cero* [WB91] (ver Sección §3.2.1).

Supongamos que w_x, representada en $x \in \mathcal{N}$, es la última palabra procesada y que la siguiente palabra identificada en el texto es w_y. La codificación de un nuevo nodo se plantea desde una perspectiva global, considerando que w_y es la primera vez que aparece en el texto. Sin embargo, la representación de una nueva arista se enfoca desde el contexto que representa el nodo x dado que lo que se pretende codificar es que la transición de w_x a w_y no había sido localizada con anterioridad en el texto. Se proponen dos funciones de codificación específicas para la representación de nuevos nodos y nuevas aristas. Ambas se describen sobre un mecanismo basado en códigos de escape y presentan las siguientes propiedades:

- La función NewVertex(w_y) se utiliza para indicar que la palabra w_y se representa en un nuevo nodo (que se añade al modelo). Esta codificación consiste en un código de escape, interpretado como "nuevo nodo", y la cadena que representa la nueva palabra. El modelo evoluciona con la inclusión del nuevo nodo y y con la actualización de $\nu(x)$ al que se añade la arista (x, y) que representa la transición de w_x a w_y, actualmente recorrida.

- La función NewEdge(y) indica la inclusión de la arista (x, y) en el vocabulario $\nu(x)$. Esta codificación consiste en un código de escape, con significado "nueva arista", y el valor entero y que representa el identificador del nodo en el que se almacena w_y.

El manejo y la representación de los códigos de escape utilizados en las funciones plantean una importante decisión (detallada en la Sección §5.3.4 junto con las diferentes alternativas para la codificación de estas funciones). Por un lado se debe considerar que, dado un alfabeto de entrada Σ formado por σ palabras diferentes, la función NewVertex se codifica en σ ocasiones. Cada una de estas codificaciones representa, implícitamente, la inclusión de una nueva arista en el modelo. Esto reduce en σ el número de veces que se codifica la función NewEdge. Dicha función, por otro lado, se codifica tantas veces como diferentes transiciones entre palabras se encuentren en el texto (menos las σ representadas con la codificación anterior). Los valores enteros codificados en la función NewEdge está comprendidos en el intervalo $[0, \sigma - 1]$, siendo más frecuentes aquellos valores x que identifiquen nodos con un alto grado de entrada $|\mathcal{D}^+(x)|$.

Codificación del texto

La función de codificación del texto se construye sobre la información estadística almacenada en las aristas del modelo. Esto supone que para codificar una determinada transición entre palabras, la arista que la representa deber haber sido previamente incluida en el vocabulario de transiciones del nodo origen. Sobre estos principios se define una tercera función, denominada Follow, para la codificación de aquellas transiciones previamente añadidas al modelo.

Consideremos que las palabras w_x y w_y, representadas en $x, y \in \mathcal{N}$, son la última palabra procesada y la siguiente palabra encontrada en el texto, respectivamente. Para poder codificar una función Follow se debe cumplir que $(x, y) \in \nu(x)$. Supongamos que dicha arista (x, y) está almacenada en la posición i-ésima de $\nu(x)$. La función Follow, simplemente, codifica el valor i con una técnica para la representación de enteros.

La información de la arista (x, y), recorrida en cada paso del algoritmo, se actualiza de acuerdo a la estrategia seguida para la gestión de ν (ver sección siguiente). Esto plantea la

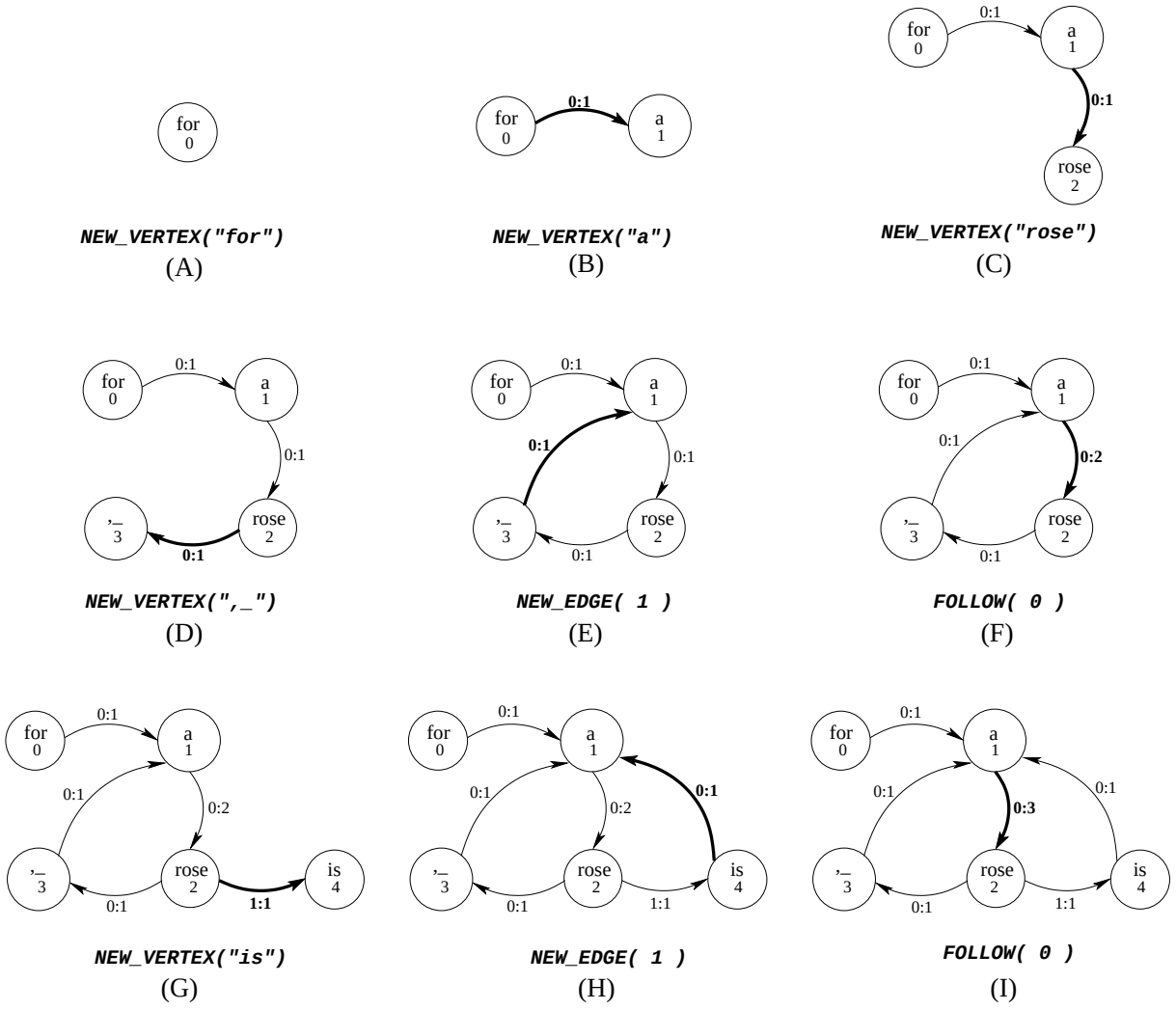


Figura 5.2: Modelado y codificación de "*for a rose, a rose is a rose*".

necesidad de reorganizar $\nu(x)$ atendiendo a la nueva configuración del vocabulario ocurrida tras la actualización de (x, y).

La figura 5.2 muestra el ejemplo de modelado y codificación de la frase "*for a rose, a rose is a rose*". Cada una de las figuras presenta, en la parte superior, el estado del modelo y en la parte inferior la función de codificación ejecutada en cada paso. La arista que representa la transición procesada en cada paso se representa con un trazo más grueso. Al igual que para el ejemplo mostrado en la figura anterior, cada arista almacena un par de valores $a : b$ que representan, respectivamente, el identificador de la arista en el vocabulario de transiciones del nodo origen y el número de veces que ésta ha sido previamente recorrida en el texto.

(A) La palabra "`for`" no ha sido previamente representada en el modelo. Un nuevo nodo (identificado por el valor 0) se añade al grafo y la función NewVertex("`for`") se codifica en el resultado comprimido. El nodo 0 se utiliza como contexto para el siguiente paso.

(B) La palabra "`a`" tampoco ha sido representada en el modelo, por lo que un nuevo nodo (1) se añade al grafo y la función NewVertex("`a`") se codifica en el resultado comprimido. Nótese cómo la arista $(0, 1)$ se añade también al modelo, almacenándose en la posición 0 de $\nu(0)$. Su contador de ocurrencias se inicializa a 1. El nodo 1 pasa a ser el contexto

utilizado para el siguiente paso.

(C)-(D) Representan pasos similares a los anteriores, añadiendo sendos nodos para la representación de las palabras "`rose`" (nodo 2) y "`,_`" (nodo 3), y codificando en el resultado comprimido sus funciones correspondientes. De la misma forma, las aristas $(1, 2)$ y $(2, 3)$ se añaden al modelo con las propiedades indicadas. El nodo 3 actúa como contexto para el paso siguiente.

(E) La palabra "`a`" está almacenada en el nodo 1 dado que ya ha sido previamente procesada. La arista $(3, 1)$ se busca en $\nu(3)$. Como no existe, la función NewEdge(1) se codifica en el resultado comprimido. Nótese que el valor entero representa el identificador del nodo destino de la arista añadida (bajo las mismas propiedades que las anteriores). El nodo 1 es ahora el contexto para el paso siguiente.

(F) La palabra "`rose`" ha sido previamente representada en el nodo 2 y en este caso la arista $(1, 2)$ si está almacenada en $\nu(1)$ bajo el identificador 0. La función Follow(0) se codifica en el fichero comprimido y la arista se actualiza (en este caso, el contador de ocurrencias pasa a ser 2). El vocabulario $\nu(1)$ se reordena aunque, en este caso, al contener una única transición, su estado permanece intacto. El nodo 2 es el nodo contexto en el paso siguiente.

(G) La palabra "`is`" no ha sido previamente representada en el modelo. Un nuevo nodo (identificado por el valor 4) se añade al grafo y la función NewVertex("`is`") se codifica en el resultado comprimido. La arista $(2, 4)$ se añade al modelo. La posición en la que esa arista se inserta en $\nu(2)$ depende de la política de gestión utilizada en cada caso. En este ejemplo se añade en la primera posición libre (en este caso la 1).

(H)-(I) La definición de la arista $(4, 1)$ y una codificación Follow (siguiendo la arista $(1, 2)$) completan los dos pasos finales.

5.3.3. Organización de ν

La función Follow es la base sobre la que se diseñan las técnicas de compresión planteadas en el capítulo actual. Como puede observarse en el ejemplo anterior, Follow puede ser entendida como una función de mapeo de aristas en posiciones de $\nu(x)$. Sin embargo, la naturaleza *adaptativa* de la etapa de modelado provoca una evolución constante de la información almacenada en el grafo de acuerdo a la actualización de la arista (x, y) recorrida en cada paso del algoritmo. Esto implica la necesidad de disponer de una estrategia de organización de ν capaz de realizar una asignación efectiva de identificadores de acuerdo a la evolución dinámica del modelo.

El proceso de actualización asociado al recorrido de las transiciones (x, y) provoca una importante sobrecarga de cómputo en la reorganización de los vocabularios $\nu(x)$ en aquellos nodos x con un alto grado de salida. La Ley de Zipf (§3.1) sugiere que la distribución de palabras en el texto es muy asimétrica. Esto supone la existencia de un pequeño conjunto de palabras (a cuyos elementos denominaremos *palabras vacías*) con una alta frecuencia de aparición mientras que el conjunto restante (formado por *palabras regulares*) se caracteriza por unas frecuencias de aparición muy bajas. Los candidatos naturales a formar parte del conjunto de palabras vacías son artículos, preposiciones, conjunciones y algunos verbos, adverbios u adjetivos [BYRN99]. Estas palabras no sólo se caracterizan por su alta frecuencia de aparición sino también por el gran número de palabras diferentes con las que se relacionan. Esta propiedad implica la existencia de un pequeño subconjunto de nodos $x \in \mathcal{N}$ con un alto grado de salida $|\mathcal{D}^-(x)|$. La actualización de $\nu(x)$ en los nodos de este subconjunto representa el cuello de botella de la propuesta actual. La utilización de una política para su gestión permite obtener un algoritmo eficiente, en tiempo y

memoria, sin que esto suponga una pérdida cuantitativa en la efectividad obtenida por la técnica de compresión. Para afrontar este problema, se consideran dos estrategias complementarias basadas el uso de heurísticas para la *ordenación* y *gestión* de ν.

Ordenación de las transiciones ν

Las estrategias para la *ordenación* ν se centran en obtener una forma eficaz de identificar las transiciones almacenadas en los vocabularios. Su objetivo principal es mantener las transiciones más *significativas* en las primeras posiciones. Esto permite obtener una distribución muy sesgada en los valores codificados mediante la función FOLLOW. Se consideran dos estrategias diferentes para la ordenación de ν:

- Basada en frecuencia. Ordena ν de acuerdo al número de veces que cada transición ha sido recorrida en el texto. Esto supone que las transiciones de cada vocabulario $\nu(x)$ siguen un orden decreciente de acuerdo a su frecuencia de aparición en el texto. Por lo tanto, esta estrategia garantiza que las primeras posiciones siempre estén ocupadas por las transiciones más frecuentes.

- Basada en la última ocurrencia ("recencia"). Considera la localidad de las transiciones apoyándose en el hecho de que las más frecuentes se distribuyen uniformemente a lo largo del texto mientras que aquellas que aparecen de forma ocasional tienden a estar concentradas en una parte concreta del mismo. De esta manera, cada vocabulario $\nu(x)$ presenta una organización cronológica de las transiciones, de forma que aquellas más recientemente seguidas aparecen en las primeras posiciones. En términos prácticos, esta estrategia mantiene en las primeras posiciones las transiciones frecuentes y aquellas que aparecen de forma concentrada en el segmento de texto procesado en cada momento.

Gestión de las transiciones en ν

Los métodos de ordenación anteriores se implementan, de forma eficiente, mediante políticas de gestión centradas en aquellos nodos con un alto grado de salida (que representan palabras vacías) dado que su reordenación compromete el rendimiento global de la propuesta. Esto conlleva establecer, en primer lugar, un mecanismo que permita diferenciar palabras regulares y vacías, de tal forma que cada uno de los grupos se gestione de manera eficiente sin que eso suponga una pérdida considerable en la información representada en el modelo.

Una palabra w_x, representada en $x \in \mathcal{N}$, se considera que es una palabra vacía si y sólo si $|\mathcal{D}^-(x)| > \alpha$. Esto supone que toda palabra cuyo grado de salida sea mayor que α pasa a ser considerada una palabra vacía. Se consideran dos políticas de gestión para todos aquellos $\nu(x)$ / $|\mathcal{D}^-(x)| > \alpha$.

- Limitación del tamaño máximo de ν. La política más sencilla es *evitar la representación de palabras vacías* obteniendo, con ello, una versión reducida del modelo original. Esta posibilidad ha sido ya probada, con un éxito limitado, en la propuesta de compresión basada en atracción léxica. Esta decisión implica limitar el tamaño máximo de $\nu(x)$ de tal forma que $\forall x \in \mathcal{N}, |\mathcal{D}^-(x)| \leq \alpha$. Esta limitación garantiza que la reordenación de los $\nu(x)$ se ejecute sobre un máximo de α transiciones.

Esta representación "reducida" del modelo requiere la utilización de una *política de reemplazo de aristas* para aquellos nodos que representan palabras relacionadas en el texto con más de α palabras diferentes. Supongamos que w_x, representada en el nodo $x \in \mathcal{N}$, es la última palabra procesada y que $|\mathcal{D}^-(x)| = \alpha$. La palabra w_y (representada en $y \in \mathcal{N}$) se identifica en el

siguiente paso del algoritmo, de forma que la transición de w_x a w_y es la procesada actualmente. Se comprueba que la transición (x, y) no existe en el modelo, de forma que ésta debe añadirse a $\nu(x)$ cuya capacidad está "completa". La política de reemplazo selecciona, para su eliminación, la arista menos significativa en $\nu(x)$ de tal forma que $|\mathcal{D}^-(x)| = \alpha - 1$ y la nueva arista (x, y) se pueda insertar.

La decisión de cuál es la arista menos significativa en un nodo depende de la política de ordenación utilizada. Las dos propuestas anteriores coinciden en que la arista menos significativa está siempre almacenada en la última posición de $\nu(x)$. La idea es común en ambos casos y trata de reemplazar aquella transición que, a priori, tiene una menor probabilidad de volver a ser recorrida de acuerdo a los criterios de ordenación seguidos. Esto supone que para la ordenación basada en frecuencia se reemplaza la transición menos frecuente en $\nu(x)$ (`LFU`, *Least-Frequently Used*) mientras que para la basada en recencia se elimina aquella transición que hace más tiempo que no ha sido recorrida en el texto (`LRU`, *Least-Recently Used*).

En términos prácticos, limitar el grado de salida de los nodos del modelo supone que algunas transiciones se insertan y eliminan del modelo en varias ocasiones. Esto puede suponer una pérdida en la efectividad considerando que que cada vez que la transición se introduzca en el modelo requiere ser recodificada de tal forma que el descompresor pueda reproducir el comportamiento del compresor.

- Diferenciación de las palabras de acuerdo a su naturaleza. Como se planteaba anteriormente, una palabra se considera vacía de acuerdo al número de palabras diferentes con las que se relaciona. Esta propiedad nos conduce a razonar que las transiciones desde una palabra vacía tienden a ser menos significativas, tanto por el elevado número de transiciones que la toman como origen como por su mayor uniformidad en la frecuencia de uso. Por lo tanto, la reordenación constante de ν, para las palabras vacías, supone un coste computacional elevado sin que ello aporte, a priori, una ventaja significativa en la efectividad de la técnica.

De acuerdo con estas propiedades, se decide mantener una reordenación dinámica de los ν asociados a las palabras regulares, definiendo dos variantes específicas para los ν de las palabras vacías:

- La variante *estática* considera de forma estricta la menor significatividad de las transiciones que parten de una palabra vacía. Esto supone que $\nu(x)$ se reordena dinámicamente hasta que se cumple que $|\mathcal{D}^-(x)| > \alpha$. A partir de ese momento, se fija la organización de $\nu(x)$ de forma que las aristas almacenadas en $[0, \alpha - 1]$ mantienen su posición hasta el final del proceso.

- La variante *dinámica* relaja la consideración anterior y asume que algunas transiciones tienen una mayor significatividad que las restantes. Desde esta perspectiva, supone que las transiciones más significativas aparecen en el grupo de las α primeras, de forma que las aristas almacenadas en $[0, \alpha - 1]$ mantienen su reordenación dinámica hasta el final del proceso.

Ambas variantes comparten el uso de una política estática para todas las transiciones almacenadas a partir de la posición α.

5.3.4. Implementación

La sección actual plantea las decisiones tomadas para la implementación del modelo. Se presentan los algoritmos de compresión y descompresión, comentando las diferentes decisiones tomadas al respecto de las estructuras de datos utilizadas así como los algoritmos genéricos considerados para la resolución de algunos problemas concretos.

Algoritmo 5.1 Compresión Edge-Guided

```
 1: N ← ∅;
 2: contexto ← INI;
 3:
 4: for all p ∈ T do
 5:   if p ∉ N then
 6:     c ← σ;
 7:
 8:     N ← N ∪ p;
 9:     ν(c) ← {NewVertex, NewEdge};
10:     arista ← (contexto, c);
11:
12:     if ν(contexto).es_vacia() then
13:       ν(contexto).gestionar();
14:     end if
15:     ν(contexto) ← ν(contexto) ∪ arista;
16:
17:     codificar(NewVertex(p));
18:   else
19:     c ← N.p;
20:
21:     arista ← (contexto, c);
22:
23:     if arista ∉ ν(contexto) then
24:       if ν(contexto).es_vacia() then
25:         ν(contexto).gestionar();
26:       end if
27:       ν(contexto) ← ν(contexto) ∪ arista;
28:
29:       codificar(NewEdge(c));
30:     else
31:       id ← ν(contexto).buscar(arista);
32:       codificar(Follow(id));
33:     end if
34:   end if
35:
36:   ν(contexto).actualizar(c);
37:   contexto ← c;
38: end for
```

El algoritmo 5.1 describe los pasos que definen la compresión `Edge-Guided` (`E-G`). El conjunto de nodos $\mathcal{N}$ se inicializa a vacío y el nodo contexto se inicializa a un valor especial INI que indica el comienzo del proceso de construcción del grafo. El texto se procesa, completamente, en el bucle `for` responsable de la lectura de una nueva palabra p en cada paso.

En cada paso del bucle se lee una nueva palabra y se comprueba si ha sido previamente representada. En caso afirmativo, se recupera la identificación del nodo que la almacena ($c \leftarrow \mathcal{N}.p$) y se busca la arista $(contexto, c)$. Este proceso de búsqueda se lleva a cabo sobre $\nu(contexto)$. Si la arista esta representada en el modelo se ejecuta la función Follow a la que se le pasa la

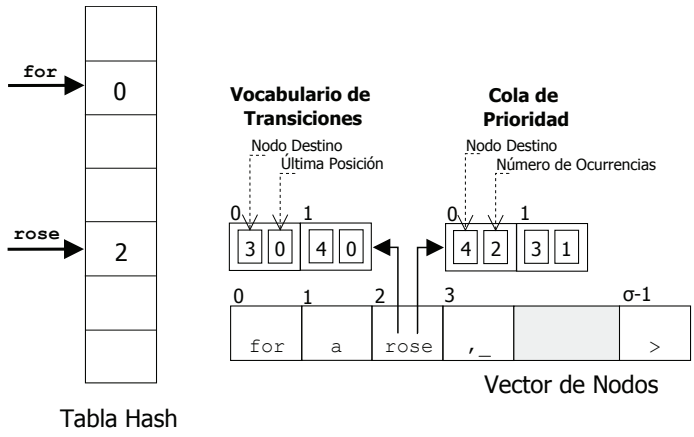


Figura 5.3: Propuesta de implementación del compresor Edge-Guided.

posición que identifica la arista en $\nu(contexto)$. Si, por el contrario, la arista no está representada en el modelo, se añade y se ejecuta la función NewEdge parametrizada con el nodo en el que se almacena la palabra destino. En este caso el identificador c del nodo destino se codifica en dicha función. Finalmente, si la palabra leída en el paso actual no ha sido previamente representada se precisa almacenarla, añadir el nodo c que la representa (inicializando c con los códigos de escape planteados anteriormente) y, finalmente, definir la arista que relaciona el nodo contexto con el nuevo nodo creado. La función NewVertex es ejecutada, codificando el evento actual así como la cadena alfanumérica que representa la palabra. Cada iteración del bucle finaliza actualizando la información almacenada en la arista recorrida y el estado $\nu(contexto)$. El nodo c pasa a ser *contexto* en la siguiente iteración.

- Implementación del compresor. La propuesta de implementación actual se centra en obtener una representación eficiente de los conjuntos de nodos y aristas teniendo en cuenta las operaciones que se realizan sobre ellos. Al igual que en el prototipo construido en el capítulo anterior, el proceso de desarrollo actual se lleva a cabo en lenguaje `C++`. La figura 5.3 muestra un ejemplo con las estructuras consideradas para la implementación del compresor.

En primer lugar se trata la implementación del **conjunto de nodos** para la que se opta por una doble estructura. Por un lado se considera el uso de una *tabla hash* sobre Σ que permita un acceso rápido a las palabras identificadas en el texto. La implementación de esta estructura (bajo las mismas propiedades comentadas en el capítulo anterior) tiene un coste espacial $O(\sigma)$. Cada celda en la tabla contiene un valor númerico i que referencia al nodo que representa la palabra en el grafo y que se almacena en la posición i-ésima del vector (`STL`) utilizado para el almacenamiento del conjunto de nodos. Cada uno de los elementos de este vector almacena tanto la palabra que representa como dos punteros a sendas estructuras que implementan el vocabulario de transiciones ν de cada uno de los nodos. Esto supone un coste $2\sigma + O(\sigma)$, para la tabla hash y el vector de nodos. Observando el ejemplo de la figura 5.3, al acceder a la tabla hash con las palabras "`for`" y "`rose`" se obtienen los valores 0 y 2 que referencian, respectivamente, sus posiciones en el vector de nodos.

La implementación del **conjunto de aristas** se lleva a cabo, de forma distribuida, en cada nodo del grafo.Cómo se citaba anteriormente, cada nodo x contiene un par de punteros que hacen referencia a las estructuras utilizadas para la implementación de $\nu(x)$. Estas estructuras se denominan *vocabulario de transiciones* y *cola de prioridad*. Ambas se implementan como vectores `STL` en los que cada elemento almacena un par de valores enteros.

El *vocabulario de transiciones* describe, exactamente, el rol de ν explicado a lo largo del

capítulo actual. Esto es, dado el nodo representado en la posición x del vector, la presente estructura permite localizar todos los nodos y tal que $(x, y) \in \mathcal{A}$. Para ello almacena un primer valor que contiene la identificación del nodo destino y un segundo que indica la posición en la que la arista se localizó, por última vez, en la *cola de prioridad* (esta decisión permite acelerar, en términos prácticos, la ejecución de la función FOLLOW). Volviendo a la figura 5.3, el vocabulario de transiciones asociado a la posición 2 del vector de nodos contiene sendas referencias a los nodos 3 y 4. Cada una de ellas representa, respectivamente, a las aristas $(2, 3)$ y $(2, 4)$ utilizadas para el modelado de las transiciones "`rose,_`" y "`rose is`". Los elementos del vocabulario de transiciones presentan un orden creciente respecto al identificador del nodo destino con el objetivo de facilitar la ejecución de procesos de búsqueda binaria sobre él.

La *cola de prioridad* plantea la organización de las aristas en el nodo de acuerdo a las estrategias de ordenación y gestión consideradas para ν. Al igual que en la estructura anterior, cada elemento en el vector contiene una referencia al nodo destino de la arista así como él número de veces que dicha arista se ha recorrido previamente en el texto. Éste es el valor utilizado para la ordenación de los elementos de esta cola de prioridad: el número de ocurrencias de cada arista[1]. En el ejemplo mostrado en la figura 5.3, el primer elemento apunta al nodo 4, dado que la arista $(2, 4)$ se ha recorrido en dos ocasiones, mientras que el segundo apunta al 3 ya que la arista $(2, 3)$ presenta una única ocurrencia previa.

Volviendo al vocabulario de transiciones, ahora podemos analizar la misión del segundo valor de cada elemento. La arista $(2, 4)$, almacenada en la posición 1, contiene una referencia a la posición 0 de la cola de prioridad que es justo en la que está actualmente ordenada dicha arista. Sin embargo, la arista $(2, 3)$ referencia la misma posición y en cambio está ordenada en la posición 1. En ambos casos, la interpretación del valor indica que la arista estaba almacenada en la posición 0 de la cola de prioridad tras su última ocurrencia en el texto. Sin embargo, la cola de prioridad se reorganiza con cada ocurrencia de la palabra "`rose`" en el texto, de tal forma que lo que este valor establece es el *límite izquierdo* a partir del cual la arista debe ser buscada en la cola de prioridad.

El coste de almacenamiento de estas estructuras se calcula de forma global. Aunque en términos prácticos el límite superior nunca se alcanza en lenguaje natural, el número máximo de posibles transiciones es σ^2 considerando que Σ contiene σ palabras diferentes. Teniendo en cuenta que la información sobre las aristas se almacena por duplicado, en el vocabulario de transiciones y en la cola de prioridad, y que para ambos casos se almacenan dos valores por arista, el coste total de estas estructuras es $O(4\sigma^2)$, considerando la representación del modelo completo. Sin embargo el grafo puede contener una versión reducida del modelo completo del texto de acuerdo a la estrategia que limita a α transiciones el tamaño máximo de ν. Esto supone,en términos prácticos, una notable reducción del coste de almacenamiento que evoluciona a $O(4\alpha\sigma)$, donde el valor de α es una fracción de σ.

- Análisis de costes del algoritmo de compresión. Una vez planteada la implementación del compresor se puede analizar el coste de cada una de las operaciones requeridas en el algoritmo. El análisis se plantea para la estrategia de ordenación basada en frecuencia.

Sobre el conjunto de nodos $\mathcal{N}$ se requieren sendas operaciones que permitan comprobar la existencia de una palabra en el grafo ($p \in \mathcal{N}$) y añadir un nodo en representación de una nueva palabra ($\mathcal{N} \cup p$). Ambas operaciones tiene un coste $O(1)$ sobre la tabla hash considerada (la inserción de un nuevo nodo también añade un elemento al final del vector de nodos).

Las operaciones sobre el conjunto de aristas $\mathcal{A}$ se implementan sobre $\nu(x)$. La operación de

[1]Nótese que para la ordenación basada en recencia este valor no sería necesario dado que dicha ordenación es siempre cronológica de tal forma que la última arista seguida desde el nodo actual siempre pasa a la primera posición de la cola de prioridad.

Operación	Comp.	Desc.
$p \in \mathcal{N}$	$O(1)$	–
$\mathcal{N} \cup p$	$O(1)$	$O(1)$
$arista \in \nu(contexto)$	$O(\log \sigma)$	–
$\nu(contexto) \cup arista$	$O(2log\sigma)$	$O(log\sigma)$
$\nu(contexto).buscar(arista)$	$O(\log \sigma + \sigma)$	–
$\nu(contexto).gestionar()$	$O(1 + \log \sigma)$	$O(\log \sigma)$
$\nu(contexto).actualizar(p)$	$O(\log \sigma)$	$O(\log \sigma)$

Tabla 5.1: Análisis de costes del algoritmo Edge-Guided.

existencia de una arista en $\nu(x)$ se lleva a cabo en tiempo $O(\log \sigma)$ dado que se puede completar mediante una búsqueda binaria en el vocabulario de transiciones. Por su parte, insertar una nueva arista en $\nu(x)$ tiene un coste $O(2 \log \sigma)$ considerando que se añade ordenadamente tanto al vocabulario de transiciones como a la cola de prioridad. En ambos casos se utiliza una búsqueda binaria para seleccionar la posición en la que nueva transición debe ser insertada. La operación `gestionar` sólo se utiliza en la estrategia de gestión que limita a α el número máximo de aristas por nodo. Tiene un coste $O(1 + \log \sigma)$, considerando que elimina el último elemento en la cola de prioridad y, posteriormente, requiere una búsqueda binaria en el vocabulario de transiciones previa a la eliminación del elemento.

Por su parte, la búsqueda de una transición en $\nu(x)$ consta de dos partes bien diferenciadas. En primer lugar busca la transición en el vocabulario de transiciones, con un coste $O(log\sigma)$, y de existir la localiza en la cola de prioridad con coste $O(\sigma)$. Esta operación, implementada de forma secuencial, utiliza como límite izquierdo el valor "última posición" que almacena la posición en la que la arista estaba almacenada, en la cola de prioridad, la última vez que se procesó. Nuestra experimentación ha demostrado que, en una gran mayoría de los casos, la arista buscada está muy próxima al límite izquierdo utilizado. Esto supone que el número de comparaciones necesarias es competitivo respecto a una eventual búsqueda binaria. Sin embargo, con la información actualmente almacenada dicha búsqueda binaria no podría ser llevada a cabo.

Finalmente la operación `actualizar`, ejecutada en cada paso del algoritmo, tiene un coste $O(\log \sigma)$ dado que requiere reinsertar en la cola de prioridad la transición procesada. La tabla 5.1 muestra un resumen de estos resultados (columna Comp.) y lo complementa con los costes que presentan estas mismas funciones en el algoritmo de descompresión (columna Desc.) que se comenta a continuación.

El algoritmo 5.2 plantea la descripción del descompresor `Edge-Guided`. Se define sobre un proceso simétrico al compresor pero con una importante diferencia: el modelo ya se encuentra representado en el texto comprimido. De esta manera, el descompresor conoce, a priori, si una palabra o una transición ha sido previamente representada en el modelo y, en este último caso, en qué posición del vocabulario de transiciones se encuentra representada. Esto supone que las funciones `palabra_representada` y `transicion_representada` no son necesarias en el proceso de descompresión. Estas propiedades facilitan notablemente el proceso de reconstrucción del modelo lo cual se traduce en una importante mejora en la eficiencia del descompresor respecto al compresor.

Haciendo un breve repaso de este algoritmo de descompresión, puede observarse como el proceso de inicialización es similar al llevado a cabo en compresión. En cada paso del algoritmo se lee un símbolo c de $\mathcal{T}_{comp}$ (en representación del texto comprimido) que actúa como indicador de la función de decodificación a utilizar. En todo los casos c contiene un número entero cuyo significado se interpreta en el vocabulario de transiciones del nodo contexto. Si $c =$ NewVertex, se decodifica una nueva palabra de Σ_{comp} y, al igual que en compresión, se inserta un nuevo nodo

Algoritmo 5.2 Descompresión Edge-Guided

```
 1: 𝒩 ← ∅;
 2: contexto ← INI;
 3:
 4: for all c ← 𝒯_comp.decode() do
 5:   if c = NewVertex then
 6:     c ← σ;
 7:     p ← Σ_comp.decode()
 8:
 9:     𝒩 ← 𝒩 ∪ p;
10:     ν(p) ← {NewVertex, NewEdge};
11:     arista ← (contexto, c);
12:
13:     if ν(contexto).es_vacia() then
14:       ν(contexto).gestionar();
15:     end if
16:     ν(contexto) ← ν(contexto) ∪ arista;
17:   else
18:     if c = NewEdge then
19:       c ← 𝒜_comp.decode();
20:       p ← 𝒩[c]
21:
22:       arista ← (contexto, c);
23:
24:       if ν(contexto).es_vacia() then
25:         ν(contexto).gestionar();
26:       end if
27:       ν(contexto) ← ν(contexto) ∪ arista;
28:     else
29:       c ← ν(contexto)[c];
30:       p ← 𝒩[c]
31:     end if
32:   end if
33:
34:   escribir_palabra(p);
35:   ν(contexto).actualizar(c);
36:   contexto ← c;
37: end for
```

y una arista desde el nodo contexto al actual. Si $c =$ NewEdge, se decodifica el siguiente símbolo en $\mathcal{T}_{comp}$ que contiene el valor que identifica el nodo *destino* en el que está representada la palabra a decodificar. Se añade una arista, con origen la palabra actual, a $\nu(contexto)$. Finalmente, si c no ha sido identificado como un símbolo de escape, estamos ante la decodificación de una función Follow. El valor leído indica la posición de la arista (en $\nu(contexto)$) que debe ser recorrida en el paso actual. En su nodo destino se encuentra la siguiente palabra a decodificar. La palabra p recuperada, en cada caso, se añade al flujo de salida y, al igual que en el compresor, se actualiza la información de la arista recorrida, se reorganiza $\nu(contexto)$ y el nodo actualmente procesado pasa a ser contexto para el siguiente paso.

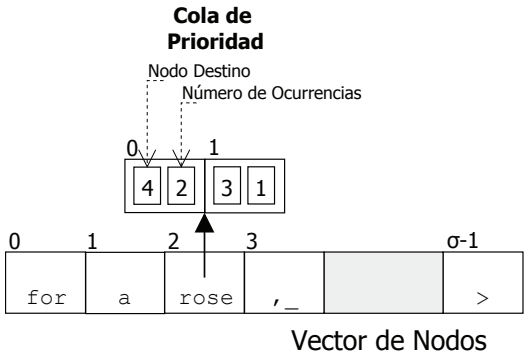


Figura 5.4: Propuesta de implementación del descompresor Edge-Guided.

- Implementación del descompresor. La implementación del descompresor se basa en la experiencia adquirida en el compresor. Este proceso plantea una simplificación del anterior considerando que el grafo se reconstruye a partir de la información suministrada en el texto comprimido. Por lo tanto, se requiere un menor almacenamiento de información en el grafo de descompresión considerando que ni las operaciones de existencia en $\mathcal{N}$ y $\nu(x)$, ni la de búsqueda en $\nu(x)$ son necesarias en este caso. La figura 5.4 muestra la propuesta de implementación para el descompresor.

Para la implementación del **conjunto de nodos** se utiliza únicamente el *vector de nodos*, con una configuración similar a la del compresor, aunque en este caso cada posición sólo almacena el puntero que referencia a la *cola de prioridad*. Su cometido es similar al desarrollado en la etapa de compresión. El coste de implementación del descompresor se reduce notablemente respecto al compresor: el vector de nodos pasa a tener un coste $O(\sigma)$ mientras que las colas de prioridad se mantienen en $O(2\sigma^2)$.

- Análisis de Costes del Algoritmo de Descompresión. El presente análisis es similar al anterior salvo porque el coste de la función de inserción de una nueva arista en ν ($\nu(contexto) \cup arista$). Su coste evoluciona a $O(\log \sigma)$ dado que las aristas , en el caso actual, sólo precisan ser añadidas a la cola de prioridad. El detalle de los costes de las operaciones, en descompresión, también se muestra en la tabla 5.1.

5.3.5. Representación de las funciones de codificación

La sección actual detalla la implementación de las funciones de codificación sobre alfabetos de salida orientados a bit y a byte. Esto da lugar, a su vez, al análisis de las políticas utilizadas en la gestión de los códigos de escape necesarios en la representación del grafo. Como resultado se obtienen sendas técnicas de compresión con un *trade-off* espacio/tiempo acorde a las propiedades derivadas del esquema de codificación.

Primero nos centraremos en las funciones para la codificación del grafo (que comparten su desarrollo sobre un mecanismo basado en códigos de escape). Esto implica la necesidad de modelar y codificar un código de escape diferente para cada una de las funciones. Ambos códigos necesitan una gestión independiente en cada uno de los nodos del grafo de tal forma que cada evento NewVertex/NewEdge se pueda codificar de acuerdo al nodo contexto.

La función NewVertex emite, en primer lugar, su código de escape (que será codificado mediante un código de enteros) y, posteriormente, la propia cadena de texto que representa la nueva palabra. Ésta cadena se añade a un flujo independiente en el que se representan todas las palabras identificadas en el texto. Este flujo (Σ_{comp}) se comprime con un algoritmo `PPM` capaz

de aprovechar las relaciones existentes entre las letras que forman parte de cada una de las palabras.

La función NewEdge, al igual que la anterior, emite su código de escape correspondiente y un segundo valor entero que representa el identificador del nodo destino de la nueva arista. Este segundo valor se añade a un flujo independiente de números enteros distribuido en el intervalo $[0, \sigma - 1]$: este flujo está compuesto por los identificadores de todos los nodos destino de las aristas representadas en el grafo. La mayoría de los valores presentan una baja frecuencia de aparición excepto aquellos que identifican los nodos en los que se representan las palabras vacías.

Finalmente, la función Follow emite el valor entero que representa la posición, en el nodo contexto, de la arista recorrida en cada paso del proceso. Estos valores se añaden al flujo principal en el que se codifican junto con los códigos de escape. Estos valores se distribuyen en el rango $[0, \sigma + 1]$, considerando que desde un determinado nodo se pueden alcanzar los σ nodos del grafo y, además, son necesarios dos códigos extra para la representación de los símbolos de escape. Sin embargo, en términos prácticos, el límite superior representa una cota irreal dado que ni siquiera las palabras vacías llegan a relacionarse con una fracción tan grande de Σ. Las propiedades de esta distribución de enteros se analizan en la sección de experimentación del capítulo actual aunque, a grandes rasgos, se caracteriza porque los primeros valores del rango presentan una frecuencia de aparición muy alta respecto al resto cuya frecuencia decrece progresivamente. Esto se debe a que las estrategias de ordenación y gestión de los vocabularios consiguen mantener las transiciones más significativas en sus primeras posiciones.

Un caso especial de lo anterior sucede al limitar el tamaño máximo de los vocabularios de transiciones. Esto supone distribuir los valores en el intervalo $[0, \alpha + 1]$ de tal forma que los primeros valores presentan unas propiedades comparables a las anteriores aumentando la frecuencia del resto al restringir el número máximo de aristas posibles en cada nodo.

Para la codificación de los dos flujos de enteros se opta por un código de longitud variable. La solución natural es utilizar un código orientado a bit que permita obtener una representación más compacta. Sin embargo, siguiendo la experiencia de las propuestas revisadas en el capítulo anterior, se considera una segunda alternativa: codificar el flujo de enteros orientado a byte y, posteriormente, aplicar un compresor universal orientado a bit sobre el resultado. Se evalúan ambas alternativas prestando especial atención a la estrategia seguida, en cada caso, para la gestión de los códigos de escape.

- Codificación orientada a bit. Utilizar un código de longitud variable orientado a bit se plantea como la solución natural para la representación del flujo de enteros resultante de la función Follow y los códigos de escape necesarios para la codificación del grafo. En términos prácticos se trabaja con un alfabeto de entrada de pequeño tamaño en el que los primeros valores acumulan la mayor parte de las ocurrencias. Esta propiedad facilita la utilización de códigos de prefijo que favorecen la asignación de palabras de código más cortas para la representación de los primeros valores del intervalo. Utilizaremos un código canónico de `Huffman` [SK64, MT97] para la codificación de este flujo.

Los códigos de escape se gestionan, de forma dinámica, en cada nodo del grafo y se codifican como si fuesen una transición más en $\nu(x)$. Para ello, cada $\nu(x)$ se inicializa con dos transiciones "virtuales" que representan los códigos NewVertex y NewEdge. La información de cada una de ellas se actualiza de manera similar a cómo se hace para el resto de transiciones. Esto es, si en un determinado paso de la etapa de modelado el nodo $x \in \mathcal{N}$ actúa como contexto y una nueva palabra o una nueva transición es encontrada en el texto, la información de la respectiva arista virtual se actualiza de la misma forma previamente explicada para las aristas regulares y $\nu(x)$ se reorganiza de acuerdo con esta actualización.

Esta política dinámica degrada ligeramente la eficiencia de la técnica en los primeros pasos

de la etapa de modelado en los que se codifica el grueso del modelo[2]. Sin embargo, este coste se amortiza en la efectividad de la técnica de codificación que es capaz de adaptarse dinámicamente a las propiedades del texto. Por un lado facilita el uso de palabras de código más cortas para representar los códigos de escape en la fase inicial de la etapa de modelado en la que los eventos de encontrar nuevas palabras y/o transiciones se distribuyen más frecuentemente. Por otro lado, favorece una codificación más óptima de las transiciones significativas de aquellos nodos con un bajo grado de salida y, por tanto, en los que los códigos de escape están infrautilizados.

Finalmente, el flujo que contiene los enteros suministrados en la codificación de las funciones NewEdge se codifica utilizando un segundo árbol de `Huffman` obtenido sobre la misma propuesta canónica anterior. En este caso, el rango de valores se distribuye, de una manera más uniforme, en el rango $[0, \sigma - 1]$. Sólo aquellos enteros que representan nodos en los que se almacenan palabras vacías, y por tanto con un alto grado de entrada, muestran una frecuencia de aparición significativamente mayor a los demás.

La implementación de la presente estrategia de codificación, sobre el conjunto de propiedades previamente definido, da lugar a la obtención de un compresor de orden 1, denominado `E-G`$_1$.

- Codificación orientada a byte. La presente estrategia toma como punto de partida los trabajos previamente referidos al respecto de la transformación del texto en una secuencia de bytes más compresible desde la perspectiva con un compresor orientado a bit. Para este caso seleccionamos el código denso `ETDC` (ver Sección §3.3.3) cuya viabilidad para este tipo de problemas ya ha sido previamente demostrada en [FNP08] y en las técnicas consideradas en el capítulo anterior.

La política de gestión de códigos de escape se simplifica profundamente para la presente estrategia al reservar las dos primeras posiciones de cada $\nu(x)$ para la representación de los códigos de escape. Esto supone que, con independencia del nodo contexto utilizado, los códigos NewVertex y NewEdge se representan siempre como `<0000000>` y `<10000001>`, respectivamente. Esta decisión elimina la necesidad de actualizar la información relativa a los códigos de escape. Este mecanismo permite mejorar ligeramente la eficiencia de la propuesta de gestión actual respecto a la anterior. A su vez, reservar estos dos primeros códigos limita a 126 el número de transiciones que se pueden codificar con un único byte. Sin embargo, esto no reduce la efectividad de la técnica, dado que la significatividad de los códigos de escape es superior, en la gran mayoría de los casos, a la de la primera transición codificada con 2 bytes.

El flujo de enteros suministrado en las funciones NewEdge se codifica con un segundo código `ETDC` y, posteriormente, se comprime con una técnica universal orientada a bit.

Cómo resultado de implementar esta estrategia de codificación se obtiene un conjunto de compresores basados en la transformación del texto sobre preprocesamiento `Edge-Guided` y su posterior compresión con una técnica universal. Estas técnicas se denominan `E-G`$_1 + X$, donde X representa la técnica universal utilizada para la compresión de las secuencias de bytes. Cómo se plantea en la sección de experimentación, esta propuesta funciona de forma efectiva con `bzip2` y `ppmdi`.

En conclusión, dado un fichero compuesto por un total de n palabras y representado sobre un grafo `Edge-Guided`, $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, el resultado de su codificación, son tres flujos complementarios:

- Un flujo principal ($\mathcal{T}_{comp}$) compuesto por n enteros de los que $|\mathcal{A}|$ son códigos de escape ($|\mathcal{N}|$ códigos NewVertex y $|\mathcal{A}| - |\mathcal{N}|$ NewEdge) y los restante $n - |\mathcal{A}|$ corresponden

[2]La identificación de nuevas palabras y/o transiciones se produce de forma esporádica en el resto de la etapa de modelado por lo que el dinamismo con el que se gestionan los códigos de escape apenas afecta a la eficiencia de la técnica.

a los enteros utilizados en las funciones FOLLOW. La efectividad de la técnica depende, principalmente, de la codificación de este flujo.

- Un flujo de caracteres (Σ_{comp}) que representan las $|\mathcal{N}|$ palabras identificadas en el texto. Este flujo se accede, en descompresión, cada vez que un código NEWVERTEX se identifica en $\mathcal{T}_{comp}$. Se codifica con `PPM` con el fin de aprovechar la relación existente entre las letras que conforman las palabras.

- Finalmente, el segundo flujo de enteros: $\mathcal{A}_{comp}$ (compuesto por los identificadores de los nodos destino de las aristas del grafo) se codifica de forma similar a $\mathcal{T}_{comp}$. Este flujo está formado por $|\mathcal{A}| - |\mathcal{N}|$ enteros (considerando que las $|\mathcal{N}|$ aristas restantes se representan implícitamente en la función NEWVERTEX).

5.4 Propuesta de orden superior sobre un alfabeto extendido

El modelo de orden 1, definido en la sección anterior, plantea una representación del texto basada en las relaciones de adyacencia existentes entre sus palabras constituyentes. Esto supone, a su vez, que el propio modelo contiene una representación estadística de todos los bigramas de palabras existentes en el texto. Esta información puede ser utilizada, desde dos puntos de vista diferentes, para complementar la representación del texto obtenida en dicho modelo:

- La información de los bigramas puede utilizarse para identificar contextos significativos de segundo orden, obteniendo una representación del texto sobre un modelo de orden 2.

- Esta misma información también se puede utilizar para la detección de frases significativas con las que extender el alfabeto de entrada original.

Ambas perspectivas sirven de base para el desarrollo de la técnica descrita en la sección actual. Sin embargo, dicha técnica no se limita a $q = 2$ y se plantea una propuesta, complementaria al modelo original, que permite obtener una caracterización estadística del texto haciendo uso de los q-gramas significativos identificados en él. La sección actual trata de demostrar como el uso de una gramática libre de contexto puede facilitar la satisfacción de los objetivos anteriores a partir de la representación jerárquica de cada uno de los q-gramas. Esta línea de investigación se desarrolla sobre la idea Gutmann y Bell [GB94] que sugiere la idea de construir un modelo de primer orden sobre un alfabeto de frases.

El resultado de esta propuesta es una técnica de compresión *semi-estática* [MPAdlFF10] que, en una primera pasada, construye una gramática del texto de acuerdo a las propiedades consideradas para la producción de reglas. Esta jerarquía de q-gramas se incorpora, en la segunda pasada, al modelo original presentado en la sección anterior de tal forma que el texto se pueda comprimir de acuerdo a la nueva representación. Para ello se necesita extender el esquema de codificación presentado en el caso base.

La sección actual se inicia con la presentación del algoritmo utilizada para la detección de q-gramas significativos. Esta explicación se lleva a cabo sobre un resumen inicial de las propiedades principales del algoritmo de `Re-Pair` [Lar99, LM00] utilizado como base para la variante utilizada. Las secciones §5.4.2 y §5.4.3 extienden las propuestas originales de modelado y codificación de tal forma que soporten la relación de jerarquía considerada. Finalmente, en la sección §5.4.4 se detallan las decisiones de implementación llevadas a cabo en la propuesta actual y que permiten obtener soluciones de orden superior, similares a las presentadas para el caso base.

5.4.1. Identificación de los q-gramas significativos

`Re-Pair` (ver Sección §3.5.2) es una técnica de compresión que utiliza una gramática libre de contexto para la construcción de un diccionario jerárquico de frases. Se plantea como una técnica *offline* respecto a su proceso de compresión, que se caracteriza por ser lento y pesado al operar sobre el texto completo. Sin embargo, este coste se compensa con la alta velocidad que presenta su proceso de descompresión, cuya eficiencia prevalece en numeros entornos.

`Re-Pair` considera un alfabeto de entrada formado por un conjunto de *primitivas* que se utilizadan en el proceso de generación de las *frases* que conforman el diccionario. Tanto primitivas como frases se refieren, genéricamente, como *símbolos*. El diccionario se representa mediante una gramática libre de contexto formada por dos tipos de reglas:

$$\mathcal{A} \rightarrow a$$

$$\mathcal{C} \rightarrow \mathcal{AB}$$

en las que a es una primitiva y $\mathcal{A}$, $\mathcal{B}$ y $\mathcal{C}$ son frases del diccionario. La construcción de la gramática sigue un proceso recursivo basado en el reemplazo de *pares activos*. Cada uno de estos pares se componen de dos símbolos adyacentes en el texto que son candidatos a ser reemplazados por una nueva regla. La única condición establecida para que un par de símbolos sea considerado *activo* es que aparezca, al menos, dos veces en el texto. La política considerada para la formación de las reglas es sencilla: en cada paso del proceso se selecciona el par activo más frecuente y se sustituye por un nuevo símbolo representativo de la nueva regla añadida a la gramática. El proceso se detiene con el procesamiento del último par activo.

Cómo resultado final del proceso se obtiene tanto la gramática $\mathcal{G}$ (*jerarquía de frases*) que se deriva del texto como la *secuencia comprimida* $\mathcal{S}$ que representa dicho texto original como una secuencia de apuntadores a reglas de $\mathcal{G}$. Cada frase en $\mathcal{G}$ está formada o por una primitiva (estas frases representarían el vocabulario original del texto: Σ) o por un par de frases que se pueden expandir en el nivel justo inferior. Esta propiedad garantiza la naturaleza jerárquica del diccionario de frases.

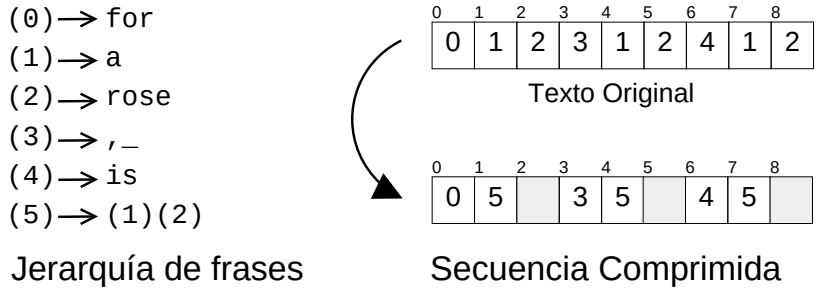


Figura 5.5: Modelado en Re-Pair de "*for a rose, a rose is a rose*".

La figura 5.5 muestra la representación de la frase "*for a rose, a rose is a rose*" sobre el modelo `Re-Pair`. En la parte superior derecha de la imagen se muestra la representación original del texto. Cómo puede observarse, cada una de las nueve posiciones de la secuencia apunta a una regla comprendida en el intervalo $[0-4]$ en el que se representan todas las palabras que componen Σ. El único par de palabras que se repite en el texto es "*a rose*", representado por las reglas 1 y 2. Esto da lugar a la nueva regla 5 que expande el par anterior. Cada aparición del par en la secuencia se sustituye por una referencia a la nueva regla, de tal forma que la posición derecha del par (posiciones sombreadas) queda "inhabilitada". La adición de la nueva

regla a la gramática permite representar el texto con 6 símbolos respecto a los 9 que componían la secuencia original

Nuestra propuesta para la identificación de los q-gramas plantea una variante del algoritmo anterior. En primer lugar se redefine el concepto de *par activo*: supongamos un texto en el que se identifican un *total* de t transiciones entre palabras, e de las cuales son transiciones *diferentes* (esto supone un grafo *Edge-Guided* formado por e aristas). El número medio de ocurrencias ($\bar{w}$) de una transición en el texto se calcula como $\bar{w} = e/t$. Consideramos como *relevantes* aquellas transiciones t_x tal que $|t_x| \geq \bar{w}$. Por tanto, todas las transiciones cuya frecuencia de aparición sea mayor o igual a $\bar{w}$ se consideran pares activos.

El proceso de generación de reglas se modifica ligeramente para el caso actual. El objetivo de esta modificación es obtener una gramática *balanceada* que asegure que los árboles de derivación de los q-gramas tengan una altura de orden logarítmico con el tamaño de Σ. Esto facilita la utilización de órdenes de k más pequeños a la hora de incorporar la jerarquía de frases al grafo `Edge-Guided` propuesto. Un sencillo mecanismo en rondas (*rounds*) garantiza el balanceo de la gramática. Para ello se restringe el uso de los símbolos generados en un determinado *round* de tal forma que sólo se puedan utilizar en los subsiguientes, pero nunca en el mismo en que ha sido obtenido [Sak05].

El algoritmo para la identificación de los q-gramas actualiza las estadísticas de e y t en cada uno de los *rounds* y, por consiguiente, el umbral $\bar{w}$ utilizado para la determinación de pares activos. El proceso finaliza cuando no queda ningún par activo en el texto o cuando se alcanza el umbral k que determina el número máximo de *rounds* considerados. El resultado obtenido es similar al comentado en `Re-Pair`: $\mathcal{G}$ representa tanto las palabras que componen Σ como el conjunto de contextos identificados en el texto, mientras que $\mathcal{S}$ contiene la representación final del texto de acuerdo a la gramática obtenida. Ambas estructuras se utilizan como entrada a la segunda etapa de nuestro algoritmo, en la que se construye un grafo `Edge-Guided` extendido para soportar la nueva jerarquía de frases.

En el presente desarrollo utilizamos una implementación propia del algoritmo `Re-Pair` adaptada a las propiedades que caracterizan la variante planteada. Dicha implementación sigue las mismas propiedades planteadas, originalmente, en la tesis de N. Jesper Larsson [Lar99]. Esto supone, en resumen, un coste en memoria de $5n + 4\sigma^2 + 4\sigma' + \lceil\sqrt{n}\rceil$ donde n es el número de palabras totales en el texto, σ el número palabras diferentes y σ' el número de símbolos diferentes (palabras + frases) en $\mathcal{G}$. En términos prácticos, tanto σ como σ' son valores muy pequeños respecto a n.

5.4.2. Modelado

La propuesta de modelado actual se centra en la representación de la jerarquía de frases en el grafo `Edge-Guided` previamente definido. Esto supone la obtención de un nuevo modelo en el que los nodos almacenen tanto las palabras, originalmente identificadas en Σ, como el conjunto de q-gramas derivados en la etapa anterior. De esta manera, las aristas representadas en el modelo de grafo actual representan las relaciones de adyacencia existentes entre secuencias de 1 o más palabras. Además, se contempla el uso de un nuevo tipo de arista cuyo objetivo es representar la *relación de jerarquía* inherente a $\mathcal{G}$ de tal forma que nos permita representar parte del árbol de derivación de cada q-grama. La etapa actual toma, como entrada, las estructuras $\mathcal{G}$ y $\mathcal{S}$ obtenidas en la etapa anterior.

De acuerdo a las propiedades que caracterizan el grafo `Edge-Guided`, cada símbolo se modela y codifica de acuerdo al símbolo que lo precede en el texto. Mientras que en $\texttt{E-G}_1$ esto supone utilizar la palabra previa como contexto, en el caso actual el símbolo precedente puede ser una secuencia de palabras de una determinada longitud. Esta propiedad caracteriza la propuesta

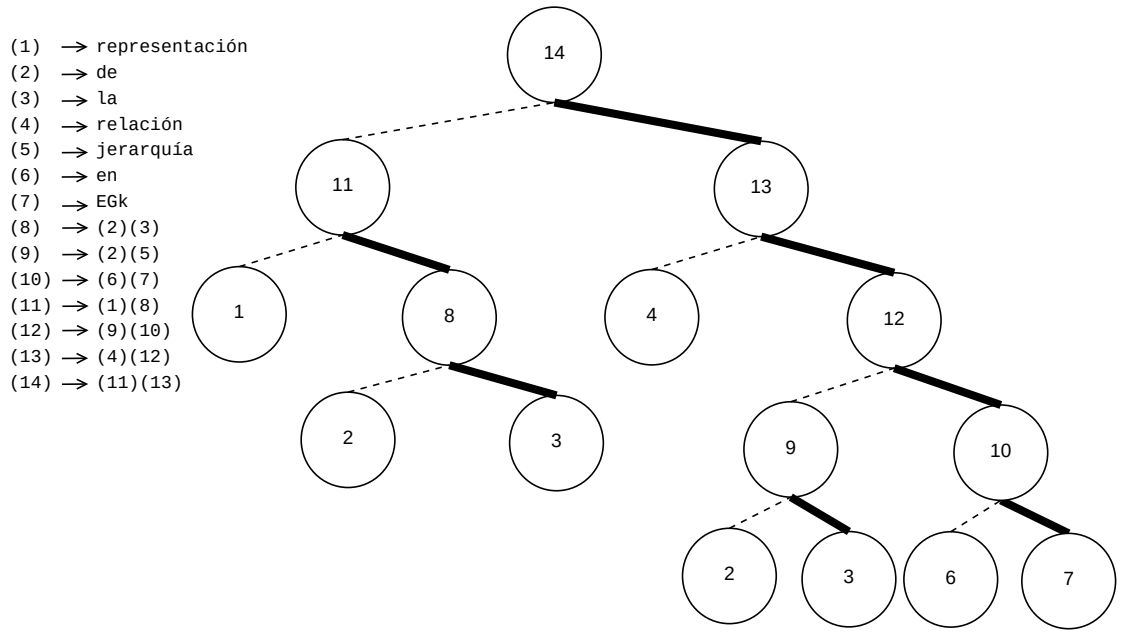


Figura 5.6: Descomposición de $\mathcal{G}$ en *heavy paths* para el q-grama "*representación de la relación de jerarquía en* EG_k".

actual como un modelo de orden 1 sobre un alfabeto de entrada extendido con un conjunto de frases significativas. En cambio, si analizamos este mismo modelo desde una perspectiva de orientación a palabras (sobre el alfabeto original Σ), se observa como un contexto formado por las k palabras precedentes se utiliza en el modelado y codificación del siguiente símbolo que, a su vez, puede ser también una secuencia de dos o más palabras.

Consideremos dos símbolos s_x y s_y, adyacentes en el texto y representados, respectivamente, en los nodos $x, y \in \mathcal{N}$. Además, supongamos que s_x es un símbolo de orden superior formado a partir de los símbolos s_{x_i} y s_{x_d} ($s_x \rightarrow s_{x_i} s_{x_d}$) representados en $x_i, x_d \in \mathcal{N}$. De acuerdo a las propiedades de nuestra propuesta, el símbolo s_y se modelaría y codificaría a través de la arista (x, y), pero supongamos que $(x, y) \notin \mathcal{A}$. Esto supondría, en el modelo original, la necesidad de codificar una función NewEdge y añadir la nueva arista al grafo. Sin embargo, en el modelo actual se puede utilizar la jerarquía que define s_x para la codificación de s_y de manera similar a como se hace en un modelo de orden superior. Para ello, sólo se necesita la expansión de la subrama derecha de s_x y utilizar su componente derecho s_{x_d} como contexto candidato para la representación de s_y. Esto facilita la posibilidad de modelar y codificar s_y a través de la arista (x_d, y), de acuerdo a un mecanismo de *blending* (explicado en la sección siguiente) que permite utilizar los diferentes componentes del árbol de derivación siguiendo su relación de jerarquía.

La representación de la relación de jerarquía de los q-gramas (de cara a satisfacer esta necesidad) adopta las propiedades que caracterizan la descomposición de una gramática libre de contexto en un conjunto de ramas disjuntas denominadas *heavy paths* [HT84]. Básicamente, esta descomposición observa $\mathcal{G}$ como un árbol de derivación formado por aristas *light* y *heavy*, de forma que las ramas obtenidas al navegar por las aristas *heavy* forman el núcleo de la representación. Para el caso actual, calificamos como *heavy* las aristas derechas de cada nodo, de manera que la *heavy path* para un determinado q-grama es aquella que permite navegar por todos los componentes derechos de su árbol de derivación, desde la raíz hasta las hoja.

La figura 5.6 muestra el árbol de derivación que genera la frase "*representación de la relación de jerarquía en* EG_k" en un determinado texto. Las líneas discontinuas indican aristas "light",

mientras que las de trazo grueso representan las aristas "heavy" y, navegando por ellas, se obtiene la descomposición de $\mathcal{G}$ en "heavy paths". El identificador de cada nodo se corresponde con la identificación, en $\mathcal{G}$, de la regla que representa. Analicemos la información contenida en el ejemplo. El nodo raíz (14) representa la regla a partir de la cual se puede obtener la frase completa. Si navegamos por la "heavy path" que nace en dicho nodo raíz, nos encontramos con el nodo 13 (del que se deriva "`relación de jerarquía en` $\texttt{EG}_k$"), el nodo 12 ("`de jerarquía en` $\texttt{EG}_k$"), el 10 ("`en` $\texttt{EG}_k$") y el 7 ("$\texttt{E-G}_k$"). Analicemos otra de las "heavy paths" representadas con el fin de generalizar las propiedades de esta representación. Por ejemplo, observemos la que tiene como raíz el nodo 11 (de la que se deriva "`representación de la`") y que continúa por 8 ("`de la`") y 3 ("`la`"). Al igual que en el caso anterior, la navegación de la "heavy path" permite extraer el componente derecho de cada q-grama $\mathcal{Q}$ de tal forma que si la codificación del siguiente símbolo no se ha podido realizar utilizando $\mathcal{Q}$ como contexto aún cabe la posibilidad de llevarla a cabo con su componente derecho $\mathcal{Q}_d$ y así hasta alcanzar el nivel de las hojas. Si llegados al final de la "heavy path" no se ha podido realizar la codificación, sólo entonces se añadirá una nueva arista al grafo.

La representación de la relación de jerarquía conlleva la necesidad de añadir una arista más a cada nodo del grafo con el fin de relacionarlo con su componente derecho en el siguiente nivel de su árbol de derivación. Nótese que como la representación de las "light paths" se descarta, de manera que la única información de $\mathcal{G}$ que se mantiene en el modelo final es aquella que representa las "heavy paths" cuya navegación representa la base para la implementación del mecanismo de *blending*.

La adición de la relación de jerarquía posibilita observar el modelo de grafo actual como un conjunto de k grafos: el de primer nivel contendría los nodos que representan las palabras que componen Σ (su configuración sería equivalente a la del grafo utilizado en el modelo original). El de segundo nivel estaría definido por aquellos nodos cuya relación de jerarquía los enlace con un nodo del primer nivel (por tanto, su componente derecho sería una palabra individual) y así sucesivamente hasta el nivel superior k, cuyos nodos estarían construidos jerárquicamente sobre el nivel $k-1$.

La figura 5.7 muestra el modelo resultante para la representación del texto "*for a rose, a rose is a rose*" sobre las características que definen la propuesta actual $\texttt{E-G}_k$. Consideremos la gramática presentada en la figura 5.5, cuyas primeras reglas representan las palabras en Σ y la última, $(5) \rightarrow (1)(2)$, deriva el bigrama "*a rose*" formado a partir de las palabras "*a*" y "*rose*". La figura actual representa dichas palabras "*a*" y "*rose*" en los nodos 1 y 2, mientras que el bigrama "*a rose*" se almacena en el nodo 3. La arista $(3, 2)$ representa, en este caso, la relación de jerarquía del bigrama con su componente derecho en el nivel inferior. El resto de aristas representadas se interpretan de forma similar que forman parte del modelo equivalente en el caso base (ver figura 5.1), excepto las representadas con trazo discontinuo. Estas últimas aristas son el resultado de ejecutar una última restricción sobre la representación jerárquica actual. La adición de una nueva arista $(x, y) \in \mathcal{A}$, con $x \in \mathcal{N}$ representando un símbolo en un orden superior k, desencadena un proceso recursivo desde el nivel $k-1$ hasta 1 en el que se añade una nueva arista (x_d, y) por nivel (siempre que $(x_d, y) \notin \mathcal{A}$), tal que la arista (x, x_d) representa la relación de jerarquía para el nodo x. Esta arista se inserta siempre al final de $\nu(x_d)$ y se inicializa con una significatividad nula dado que no ha sido realmente utilizada. Como puede verse en la figura anterior, las aristas $(2, 4)$ y $(2, 5)$ se crean con 0 ocurrencias dado que las transiciones "*rose is*" y "*rose,* " que representan no han sido directamente recorridas en el texto. Por ello, calificamos estas aristas como "virtuales". En el momento que las transiciones que representan se identifiquen por primera vez en el texto, estas aristas evolucionarán a un estado "regular" y se inicializarán con su primera ocurrencia. La utilización de estas aristas virtuales provoca, en términos prácticos, un ligero aumento en la cantidad de memoria requerida para la representación

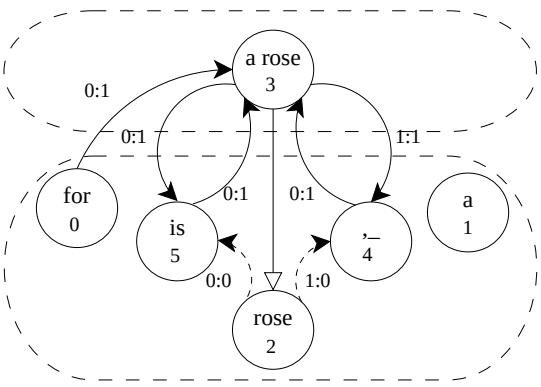


Figura 5.7: Modelado en Edge-Guided$_k$ de "*for a rose, a rose is a rose*".

del modelo. Sin embargo, al haber sido previamente representadas, su primera ocurrencia no precisa de la codificación de una función NewEdge sino que utiliza una codificación Follow cuyo coste, en bits, tiende a ser inferior en todos los casos.

Finalmente, los óvalos mostrados con trazo discontinuo delimitan los 2 grafos que se pueden identificar en el modelo actual de acuerdo al orden de los símbolos que representan. El primer nivel está formado por todos aquellos nodos que representan palabras (este grafo de orden 1 plantearía una relación equivalente a la mostrada en figura 5.1 aunque el conjunto de aristas es diferente al anterior dado que la relación de adyacencia, en el caso actual, considera un alfabeto de entrada extendido), mientras que el segundo nivel contiene el nodo que representa el bigrama "*a rose*" que está relacionado jerárquicamente con el nodo 2 en el primer nivel.

5.4.3. Codificación

La extensión del modelo de grafo original, con la relación de jerarquía introducida a través de la gramática, plantea la necesidad de disponer de una nueva función NewContext que permita su representación sobre una extensición del esquema de codificación original (ver §5.3.2).

La codificación del grafo requiere, en este caso, plantear una representación de $\mathcal{G}$ que permita su reconstrucción en el descompresor. Por un lado, las reglas del tipo $\mathcal{A} \rightarrow a$ representan las palabras en Σ de tal forma que su codificación se lleva a cabo, al igual que en el caso anterior, a través de la función NewVertex. La nueva función NewContext se utiliza para la representación de las reglas de tipo $\mathcal{C} \rightarrow \mathcal{AB}$. Por lo tanto, esta función se responsabiliza de la representación individual de cada uno de los componentes de la regla atendiendo tanto a la naturaleza particular del símbolo que representa cada componente como a su estado actual en el modelo:

(a) *El componente es una palabra y no ha sido previamente representada en el modelo.* La codificación de la función NewVertex indicaría la inserción de un nuevo nodo en el modelo para su representación. Nótese que la codificación de NewVertex, en el ámbito de NewContext, no contempla la inserción de una nueva arista.

(b) *El componente es un símbolo de orden superior y no ha sido previamente representado en el modelo.* El anidamiento de una nueva función NewContext indica esta situación. Esto supone que la codificación de funciones NewContext puede llevarse a cabo de forma recursiva de acuerdo al árbol de derivación de cada regla.

(c) Finalmente, con independencia de su orden, *si el símbolo ya ha sido previamente representado en el modelo* se codifica con una función NewEdge. La interpretación actual, de esta

función, varía respecto a su caracterización original. En este caso su único cometido es referenciar el identificador del nodo en el que está representado el símbolo componente.

A continuación se muestra un ejemplo real que cubre las posibilidades comentadas a la hora de codificar una función NewContext. Supongamos que "*Tokyo Stock Exchange*" ha sido identificado como un trigrama significativo en el texto y su árbol de derivación se construye de acuerdo a las siguientes reglas:

$$\mathcal{G}: \begin{cases} (1) \rightarrow Tokyo \\ (2) \rightarrow Stock \\ (3) \rightarrow Exchange \\ (4) \rightarrow (1)(2) \\ (5) \rightarrow (4)(3) \end{cases}$$

La primera ocurrencia de la secuencia "*Tokyo Stock Exchange*" se codifica de diferente forma de acuerdo a los siguientes supuestos:

- Las reglas $(1), (2), (3)$, de las que se derivan las palabras consideradas, no han sido previamente procesadas. El trigrama se codifica como:

  `NewContext(NewContext(NewWord('Tokyo'),NewWord('Stock')),NewWord('Exchange'))`

- Alguna de las reglas de las que se deriva una palabra, por ejemplo (2), no ha sido previamente procesada (el razonamiento sería similar si (1) o (3) no estuviesen almacenadas en el modelo) mientras que las palabras derivadas de (1) y (3) se almacenan, respectivamente, en los nodos $1, 3 \in \mathcal{N}$. El trigrama se codifica como:

  `NewContext(NewContext(NewEdge(1), NewWord('Stock')), NewEdge(3))`

- Las reglas $(1), (2), (3)$ están representadas en el modelo (supongamos que cada una de las palabras está almacenada, respectivamente, en los nodos $1, 2, 3 \in \mathcal{N}$) pero la regla (4), de la que se deriva el bigrama "*Tokyo Stock*", no ha sido previamente representada. El trigrama se codifica como:

  `NewContext(NewContext(NewEdge(1), NewEdge(2)), NewEdge(3))`

- Todas las reglas, desde la (1) hasta la (4), están representadas en el modelo (supongamos que las palabras se almacenan en los mismos nodos que en el caso anterior mientras que el bigrama se almacena en el nodo $4 \in \mathcal{N}$). El trigrama se codifica como:

  `NewContext(NewEdge(4),NewEdge(3))`

Al igual que el resto de funciones utilizadas para la codificación del grafo, la representación de NewContext está basada el mecanismo de códigos de escape, previamente explicado. Esto conlleva la necesidad de gestionar un tercer código de escape.

Por su parte, la codificación del texto de acuerdo al nuevo modelo mantiene las propiedades de Follow comentadas en el caso base. Sin embargo, se debe considerar que la información propuesta en $\texttt{E-G}_k$ soporta la implementación de un mecanismo de *blending* que permite aprovechar la representación jerárquica de los q-gramas considerados en el modelo.

Algoritmo 5.3 Mecanismo de blending en compresión

Require: *arista*
1: $encontrada \leftarrow false$;
2: $posicion \leftarrow 0$;
3:
4: **while** *no_encontrada* **do**
5: **if** $arista.destino \notin \nu(arista.origen)$ **then**
6: **if** $es\_hoja(arista.destino)$ **then**
7: $posicion \leftarrow NO\_ENCONTRADA$;
8: $encontrada \leftarrow true$;
9: **else**
10: $posicion \leftarrow posicion + |\nu(arista.origen)|$;
11: $arista.origen \leftarrow jerarquia(arista.origen)$;
12: **end if**
13: **else**
14: $posicion \leftarrow posicion + \nu(contexto).buscar(arista)$;
15: $encontrada \leftarrow true$;
16: **end if**
17: **end while**
18:
19: **return** posicion;

Mecanismo de *blending*. El presente mecanismo facilita la utilización del árbol de derivación de un símbolo de orden superior a la hora de codificar las transiciones que parten desde él. Este mecanismo se integra en la función que evalúa la existencia de una transición en un determinado $\nu(x)$ (línea 23 del algoritmo 5.1). El algoritmo 5.3 describe el procedimiento que implementa este mecanismo.

El mecanismo de *blending* se utiliza en la búsqueda de aristas que tienen como origen un nodo que representa un símbolo de orden superior. Dada la transición $s_x s_y$, cuyos símbolos se almacenan en los nodos $x, y \in \mathcal{N}$, suponemos que s_x es un símbolo de orden superior cuya relación de jerarquía lo une con el símbolo s_{x_d}, representado en $x_d \in \mathcal{N}$, en el nivel justo inferior. El procedimiento utiliza una variable (`encontrada`) para la señalización de que la transición se encuentra en el modelo. Se inicializa a `false`, mientras que la variable `posicion` almacena el valor 0. El bucle representa la navegación de la "heavy path" asociada s_x hasta que se localiza la arista solicitada o hasta alcanzar la hoja final.

Cada iteración comprueba la existencia de la arista en el vocabulario de transiciones del nodo origen (ν`(arista.origen)`). De no existir, se comprueba que dicho nodo origen no es la hoja de la "heavy path". Si lo fuese, tendríamos la seguridad de que la arista no puede ser representada de acuerdo al estado actual del modelo, por lo que sería necesaria la codificación de una nueva función NewEdge parametrizada con el valor `arista.destino`. Si, por el contrario, el nodo no es la hoja, entonces podemos seguir navegando por la "heavy path". Como puede observarse en la línea 10, el valor de la variable `posicion` se incrementa de acuerdo al número de elementos almacenados en ν`(arista.origen)`. La operación `jerarquía` devuelve el siguiente nodo en ella que pasa a ser utilizado como nodo origen en la siguiente iteración del bucle.

Si la arista solicitada existe en ν`(arista.origen)`, entonces la transición $s_x s_y$ se puede codificar utilizando el nodo actual como origen de la transición. La variable `posicion` se incrementa con el identificador de la transición en ν`(arista.origen)`. El valor final, de dicha variable `posicion`, acumula la suma de los tamaños de los ν de todos los nodos procesados en la

Algoritmo 5.4 Mecanismo de blending en descompresión

Require: $contexto, c$
1: **while** $|\nu(contexto)| > c$ **do**
2: $c \leftarrow c - |\nu(contexto)|$;
3: $contexto \leftarrow jerarquia(contexto)$;
4: **end while**
5:
6: **return** $\nu(contexto)[c]$;

"heavy path" de s_x más la posición en la que se localiza la arista actual. Nótese que si la arista se localiza directamente en x, el valor de la variable es la posición que la arista ocupa en $\nu(x)$.

Supongamos que la transición actual se ha localizado en el j-ésimo nodo de la "heavy path" de x. Esto supone que la función FOLLOW emitida está parametriza con el valor de la variable `posicion`, obtenido como: $posicion = \sum_{i=j+1}^{k} |\nu(x_{d_i})| + \nu(x_{d_j}).buscar(arista)$, donde x_{d_j} es el nodo en el que se localiza la arista y $|x_{d_i}|$ representa el tamaño de ν para cada uno de los nodos visitados en el recorrido de la "heavy path". Esta decisión permite ejecutar el mecanismo de *blending* sin la necesidad de utilizar códigos de escape adicionales para indicar la transición de un nodo a otro en la "heavy path".

La utilización del mecanismo de *blending* supone una pequeña variación en el algoritmo de descompresión que debe ser capaz de imitar el comportamiento del compresor a la hora de ejecutar dicho mecanismo. Esta modificación se incorpora a la hora de acceder a una determinada arista identificada mediante la función FOLLOW (línea 29 del algoritmo 5.2). La decodificación de FOLLOW devuelve un valor entero c que ahora precisa ser comprobado en $\nu(contexto)$. Si se cumple que $c \leq \nu(contexto)$, entonces se tiene la seguridad de que no se ha utilizado el mecanismo de *blending* en la codificación y el nodo destino de la arista se recupera en la c-ésima posición de $\nu(contexto)$. Sin embargo, si $c > \nu(contexto)$ significa que la arista ha sido codificada a lo largo de la "heavy path" del nodo contexto. Esto requiere la navegación de la misma (a través de la función `jerarquía` que devuelve el siguiente nodo en la "heavy path"), decrementando c de acuerdo al tamaño de ν en cada iteración. El bucle se detiene en el primer nodo en el que se cumple que $c \leq \nu(contexto)$, recuperando la información del c-ésimo elemento en el vocabulario ν del nodo actualmente visitado.

5.4.4. Implementación

La implementación del modelo actual, y de los correspondientes algoritmos adaptados a sus necesidades, utiliza la experiencia adquirida en el caso base. El algoritmo de compresión es similar al presentado para $\texttt{E-G}_1$ excepto por la necesidad de implementar el mecanismo de *blending* y porque su inicio toma como entrada las estructuras $\mathcal{G}$ y $\mathcal{S}$ obtenidas como resultado de la ejecución de la etapa inicial de detección de los q-gramas significativos.

El proceso de compresión actual no hace una segunda lectura del texto sino que utiliza $\mathcal{S}$ para la detección del siguiente símbolo a codificar. Esto facilita conocer si el símbolo ha sido ya representado sin más que mantener un apuntador desde la regla que lo representa en $\mathcal{G}$ a su posición en el vector de nodos. Si esta referencia es nula, el símbolo no está representado y será necesaria la codificación de una función NEWVERTEX (si es una regla del tipo $\mathcal{A} \rightarrow a$) o NEWCONTEXT (si la regla es del tipo $\mathcal{A} \rightarrow \mathcal{BC}$).

La implementación de $\mathcal{G}$ considera dos estructuras complementarias para la representación de los dos tipos de reglas. Por un lado, las reglas $\mathcal{A} \rightarrow a$ se almacenan sobre un *vector de primitivas* que almacena las σ reglas de las que se derivan todas las palabras diferentes identificadas en el texto. Cada elemento contiene un puntero a la posición del vector de nodos en la que se

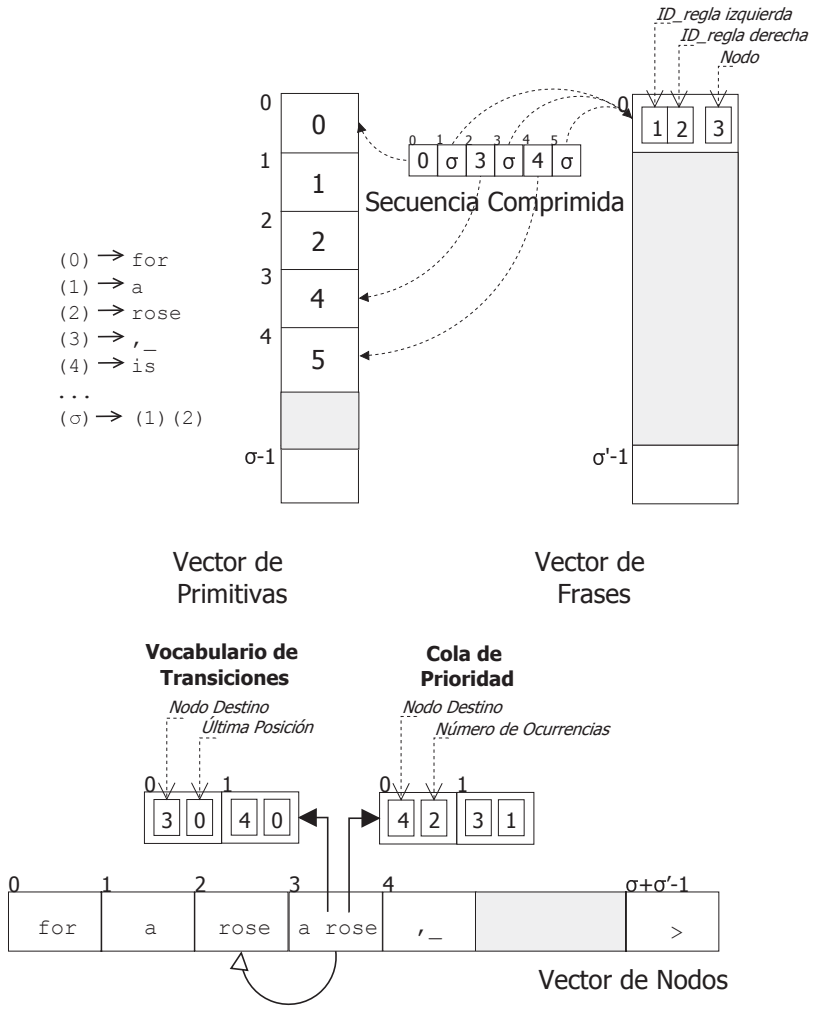


Figura 5.8: Propuesta de implementación del compresor Edge-Guided$_k$.

almacena la palabra. Por otro lado, $\mathcal{A} \to \mathcal{BC}$ se almacenan en un segundo *vector de frases* que almacena las σ' reglas del tipo actual. Cada una de las posiciones del vector contiene tres valores enteros: dos de ellos representan los identificadores, en la gramática, de las reglas izquierda y derecha que lo componen, mientras que el tercero apunta a la posición del vector de nodos en la que el q-grama está representado. Esto supone un coste $\sigma + 3\sigma'$ para el almacenamiento de la gramática.

La parte superior de la figura 5.8 muestra la configuración de estas estructuras para la gramática de ejemplo mostrada en la zona izquierda de la misma figura. Como puede observarse, las 5 reglas de las que se derivan palabras se almacenan en las primeras posiciones del *vector de primitivas*. Nótese como cada regla se identifica por su posición en dicho vector. Por ejemplo la regla $(2) \to rose$ se representa en la posición 2 del vector. Su contenido mantiene una referencia a la posición 2 del vector de nodos en la que se almacena el nodo que representa la palabra *rose*.

Por otro lado, la regla $(\sigma) \to (1)(2)$ se almacena en la primera posición del *vector de frases*. Al igual que en el caso anterior, estas reglas se identifican por su posición. Observemos dicho *vector de frases*: su posición 0 almacena la regla σ, mientras que sus componentes son las reglas 1 y 2, de las que se derivan las palabras "*a*" y "*rose*"; el valor 3 indica la posición, del vector de nodos, en la que se representa el presente bigrama. El valor -1 (para el ID de un nodo) indica, en los *vectores de primitivas* y *de frases* que la regla no ha sido aún representada en el grafo.

Por su parte, la secuencia comprimida $\mathcal{S}$ se implementa sobre un vector de k posiciones que contiene el conjunto de símbolos que permite reconstruir el texto original de acuerdo a la configuración final de $\mathcal{G}$. Cada posición almacena un entero que referencia la regla de la que se deriva el siguiente símbolo a procesar.

Finalmente,el vector de nodos mantiene su estructura original con la excepción de aquellas posiciones que representan símbolos de orden superior. En ellas se almacena un puntero complementario que representa la relación de jerarquía. En la figura puede verse como el nodo 3, que representa el bigrama "*a rose*", referencia el nodo 2 que contiene su componente derecho "*rose*". Por tanto, el coste de esta estructura es ligeramente superior al del caso anterior: $2\sigma+3\sigma'+O(\sigma)$.

La implementación del conjunto de aristas, sigue las mismas propiedades consideradas para el caso base. Esto supone un coste total $O(4(\sigma+\sigma')^2)$.

Por su parte, el algoritmo de descompresión se modifica, exclusivamente, para la implementación del mecanismo de *blending*. Las estructuras utilizadas son similares a las consideradas para $\texttt{E-G}_1$ con la excepción de las posiciones del vector de nodos que almacenan símbolos de orden superior ya que éstas necesitan representar el puntero que implementa la relación de jerarquía. Nótese que no es necesario disponer explícitamente de una representación de $\mathcal{G}$ dado que la información necesaria para la reconstrucción de los árboles de derivación se obtiene mediante la interpretación de las funciones NewContext.

El coste de las operaciones actuales es similar al mostrado en $\texttt{E-G}_1$ excepto en aquellos casos en los que se utiliza la relación de jerarquía. La inclusión de un nuevo nodo en el grafo $(\mathcal{N} \cup p)$ pasa a tener un coste ligeramente superior $O(1+2(k-1))$, donde k es el orden del símbolo. Por otro lado, el coste de las operaciones ejecutadas sobre $\nu(x)$ evoluciona considerando la necesidad de implementar el mecanismo de *blending*; la operación $arista \in \nu(contexto)$ pasa a tener un coste $O(k\log(\sigma+\sigma')+(\sigma+\sigma')))$. En lo que respecta a la descompresión, la decodificación de una función Follow pasa de un coste $O(1)$ a un coste $O(k)$ para aquellos casos en los que se necesario navegar por el árbol de derivación.

Representación de las funciones de codificación. El modelo actual soporta las mismas estrategias de codificación presentadas en el caso base. Esto supone mantener sendas propuestas *orientada a bit* y *orientada a byte*. Ambas utilizan el mismo mecanismo diseñado, originalmente, para la gestión y representación de los códigos de escape.

Esto es, la estrategia *orientada a bit* maneja de forma dinámica la información estadística asociada al nuevo código de escape, añadiendo una tercera arista "virtual" cuya codificación se adapta, progresivamente, al procesamiento del texto. Esta implementación da lugar a una técnica de compresión denominada $E-G_k$. Por otro lado, la estrategia *orientada a byte* fija la representación de NewContext mediante el código `ETDC <10000010>`, eliminando la necesidad de actualizar dinámicamente su información estadística. Al igual que en el caso base, esta estrategia de implementación permite obtener una familia de compresores $E-G_k+X$ donde X representa la técnica universal utilizada para la compresión de la secuencia de bytes obtenida.

5.5 Experimentación

La sección actual revisa los resultados experimentales obtenidos con las diferentes técnicas *Edge-Guided* planteadas en este capítulo. La figura 5.5 clasifica las técnicas estudiadas en la presente experimentación. Como puede observarse, el grueso de la experimentación se focaliza en el caso base: $\texttt{E-G}_1$. Sobre el modelo orientado a palabras se estudia la influencia de los parámetros considerados en la propuesta actual así como las diferentes estrategias de gestión de los vocabularios de transiciones propuestas. Esta experiencia se utiliza como base para la selección de los parámetros que obtienen un mejor *trade-off* espacio/tiempo y se aplican sobre el modelo $\texttt{E-G}_k$

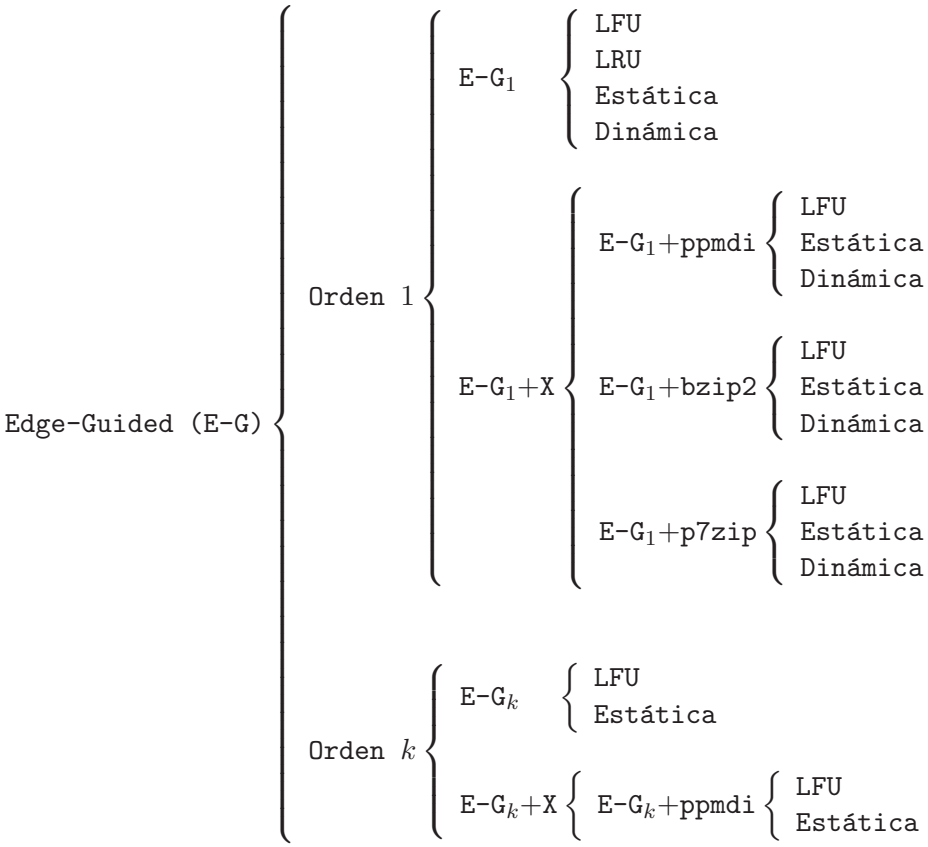


Figura 5.9: Clasificación de las técnicas estudiadas en la experimentación actual.

extendido con los q-gramas significativos. La sección finaliza con una evaluación de los resultados obtenidos por nuestras propuestas respecto a otras técnicas comparables dentro del contexto de la compresión de texto.

La descripción de los recursos utilizados en este proceso de experimentación están completamente detallados en el apéndice A.

5.5.1. Terminología

Las técnicas propuestas se refieren, genéricamente, como E-G$_1$ y E-G$_k$. En este último caso, el subíndice k indica el orden máximo de los q-gramas con los que se extiende el alfabeto original. De esta manera, los q-gramas de mayor orden considerados para E-G$_k$ están formados por la combinación de dos símbolos de orden $k-1$. Esto supone que el proceso de identificación de q-gramas se lleva a cabo en $k-1$ *rounds*.

Sobre las técnicas propuestas se obtienen diferentes variantes de acuerdo a la gestión de ν (ver §5.3.3). En ambas se utiliza el valor α como umbral entre las palabras consideradas regulares y vacías. Por un lado, se considera la limitación del tamaño máximo de los vocabularios de transiciones y el uso de una política de reemplazo sobre ellos. De acuerdo con esta estrategia de ordenación/reemplazo se obtienen sendas variantes referidas como LFU (estrategia basada en frecuencia) y LRU (estrategia basada en recencia). En este caso, el valor α se interpreta como el tamaño máximo del vocabulario de transiciones.

Por otro lado, se consideran políticas de gestión diferentes para los vocabularios de los nodos que representan palabras regulares y aquellos que almacenan palabras vacías. La clasificación de cada uno de los nodos se lleva a cabo sobre α. De acuerdo a las políticas de ordenación,

utilizadas en este caso, se obtienen sendas variantes `estática` y `dinámica`.

Las variantes `LFU`, `estática` y `dinámica` convergen en la misma política de gestión de ν al fijar un valor $\alpha = \infty$. Por un lado esto supone no limitar el grado de salida máximo de un nodo (`LFU`) y por el otro considerar todas las palabras como regulares (`estática` y `dinámica`). Los resultados presentados bajo la restricción $\alpha = \infty$ son comunes para estas tres variantes.

5.5.2. Resultados experimentales para E-G$_1$

En primer lugar estudiaremos las propiedades que definen el modelo utilizado en el caso base. Analizaremos, principalmente, la relación entre el número de nodos y aristas diferentes que forman el grafo obtenido como modelo. Estos mismos números se utilizan como base para el estudio de los grafos obtenidos al limitar el tamaño máximo de ν. Esto permite, a su vez, analizar la bondad de las técnicas de reemplazo consideradas.

Texto	**Nodos** $\|N\|$	**Aristas** $\|A\|$	**Palabras**	$\|A\|/\|N\|$
`WSJ010`	53832	573453	2217389	10, 65
`CR`	117718	1609786	10113132	13, 67
`WSJ100`	149658	2856555	22161021	19, 09
`FT94`	295018	4400311	43039675	14, 91

Tabla 5.2: Tamaño de los modelos `Edge-Guided` para textos de diferentes tamaños

La tabla 5.2 compara el tamaño de los modelos `Edge-Guided` obtenidos para la representación de textos de diferente tamaño. Las dos columnas centrales contabilizan el número de nodos y aristas que forman cada uno de los grafos, mientras que la cuarta columna cuantifica el tamaño del texto a partir del número total de palabras identificadas en él. La columna *aristas* sugiere el mejor estimador posible para el análisis del tamaño del modelo y, por consiguiente, de su eficiencia en una implementación funcional. El tamaño del conjunto de aristas representa la suma total del número de transiciones almacenadas, de forma distribuida, en los vocabularios de cada uno de los nodos del modelo. Esto nos permite estimar, de forma más realista, el coste espacial de almacenar el conjunto de aristas en $\mathcal{G}$. Cómo se planteaba en §5.3.4, esto supone costes $O(4\sigma^2)$, para el compresor, y $O(2\sigma^2)$, para el descompresor. Sin embargo, en términos prácticos, este coste es mucho menor como puede observarse en el cociente $|A|/|N|$ mostrado en la quinta columna. La relación entre el número de nodos, $|\mathcal{N}|$, y el número de aristas, $|\mathcal{A}|$, posee una naturaleza lineal $|\mathcal{N}| = m|\mathcal{A}|$, donde m es propiedad específica de cada texto. Para los ejemplos analizados $m \in [10, 20] < \sigma$, lo que transforma el coste cuadrático sobre σ en un coste lineal. De esta forma, se puede considerar que almacenar el conjunto de aristas tiene un coste $O(4m\sigma)$ y $O(2m\sigma)$ para compresor y descompresor respectivamente. Estos mismos órdenes de complejidad caracterizan la versión reducida del modelo obtenida al limitar a α elementos el tamaño máximo de ν ya que, en términos prácticos, $\alpha > m$.

Sin embargo la elección de α tiene un impacto importante tanto en la eficiencia como en la efectividad de la técnica. En primer lugar, podemos analizar que el coste espacial de aumentar el valor de α no es muy elevado respecto al coste medio de almacenamiento del modelo en memoria, dado que este valor sólo afecta a aquellos nodos que representan palabras vacías (a priori, una pequeña proporción del total). Sin embargo, la elección de un determinado valor de α puede tener una importancia significativa en los tiempos de cómputo. Las palabras consideradas vacías, a tenor del valor de α seleccionado, poseen una alta frecuencia de aparición en el texto lo que supone la necesidad de llevar a cabo procesos constantes de reemplazo y reordenación de sus vocabularios de transiciones, cuyo tamaño será, en todo momento, igual a α. Esto repercute directamente sobre la operación `actualizar`, responsable de la actualización del nodo que interviene como origen en cada arista y cuyo coste es $O(log\alpha)$. Esto supone que al pasar de un valor $\alpha_1 = 2^x$ a $\alpha_2 = 2^{x+1}$, el coste medio de la operación se incrementa en una unidad (pasa

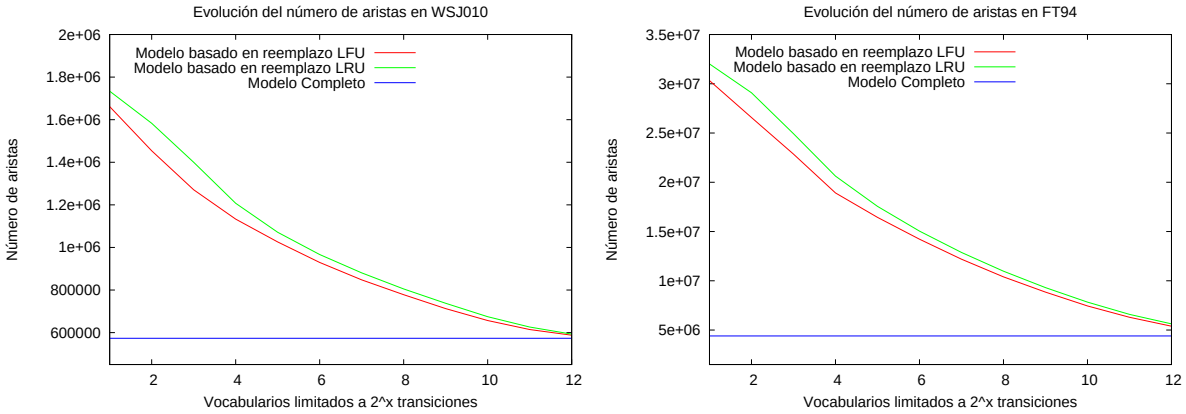


Figura 5.10: Evolución del tamaño del grafo sobre vocabularios de transiciones limitados a $\alpha = 2^x$.

Texto	Reemp.	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}
WSJ010	LFU	71,11 %	59,95 %	49,46 %	40,69 %	33,16 %	26,28 %	20,45 %	15,70 %	11,42 %	7,59 %	4,37 %	1,99 %
	LRU	74,38 %	65,77 %	55,28 %	44,05 %	35,20 %	27,96 %	21,92 %	16,92 %	12,55 %	8,40 %	4,90 %	2,22 %
FT94	LFU	69,48 %	60,38 %	51,23 %	41,39 %	34,82 %	28,72 %	23,02 %	17,84 %	13,36 %	9,42 %	6,23 %	3,70 %
	LRU	73,41 %	66,22 %	56,05 %	45,37 %	37,41 %	30,64 %	24,60 %	19,18 %	14,43 %	10,32 %	6,92 %	4,25 %

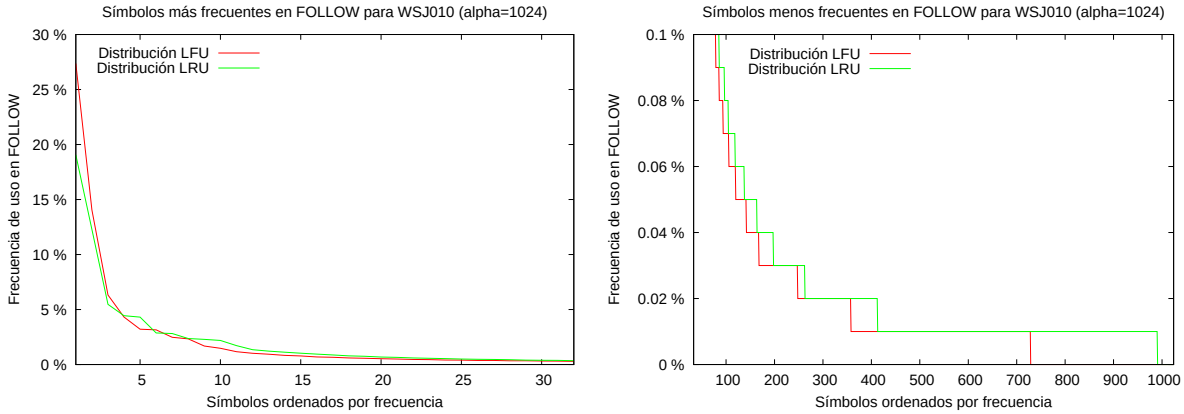
Tabla 5.3: Porcentaje de aristas reemplazadas en los grafos construidos para los textos anteriores.

de x a $x+1$). Cómo se verá más adelante, en términos prácticos este coste es poco significativo para valores pequeños de α. Sin embargo, para aquellos α para los que la técnica se demuestra competitiva es necesario tener en cuenta el contexto de ejecución a la hora de seleccionar el valor de α a utilizar y el *trade-off* espacio/tiempo obtenido con este parámetro.

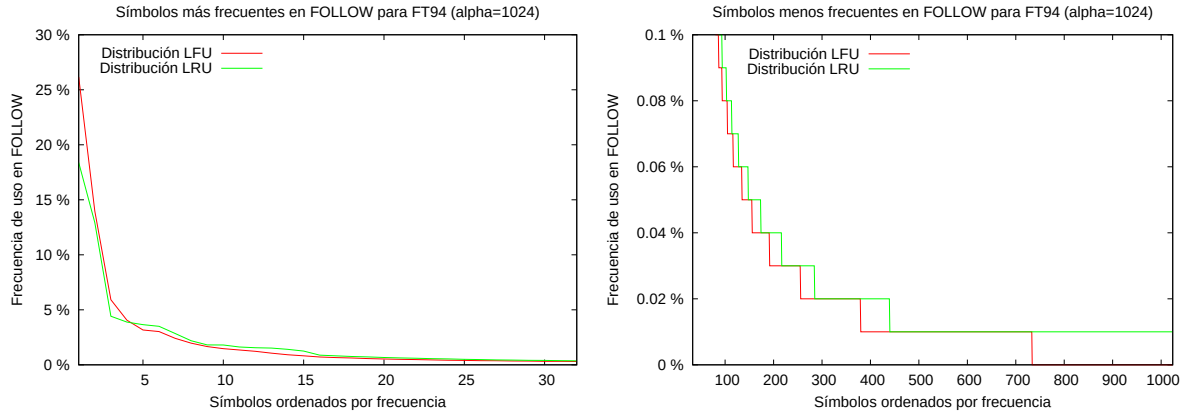
La elección de α tiene también un impacto importante en la efectividad de la técnica sobre las estrategias `LFU` y `LRU`. Esto se debe a que algunas aristas se pueden representar en múltiples ocasiones al haber sido previamente reemplazadas del modelo y, posteriormente, ser requeridas al encontrar en el texto la transición que representan. La figura 5.10 muestra la evolución del tamaño del grafo (de acuerdo a $|\mathcal{A}|$) obtenido para los textos `WSJ010` (izquierda) y `FT94` (derecha) al considerar valores crecientes de α. La tabla 5.3 complementa esta representación con el porcentaje de aristas reemplazadas, en los textos anteriores, para los valores de α considerados.

La recta *modelo completo* indica, para cada uno de los textos, el número de aristas representadas en el grafo cuando no se establece limitación en el tamaño máximo de ν. Este valor representa el límite inferior de $|\mathcal{A}|$ y, por consiguiente, el número mínimo de codificaciones NewEdge que son necesarias para representar la configuración del grafo sobre el que se modela el texto. Por su parte, las otras dos curvas muestran la evolución de $|\mathcal{A}|$ al limitar el tamaño máximo de ν ($\alpha = 2^x$). El número de aristas codificadas se reduce progresivamente al aumentar el tamaño de α (a partir de 2^{10} la tasa de reemplazos ya es inferior al 10 % del número total de aristas representadas). Sin embargo, el número de reemplazos es siempre mayor para la estrategia `LRU` (hasta un 13 % más para la colección `FT94`) lo que supone, a priori, una menor efectividad dado que el número de aristas representadas es siempre mayor. Esta propiedad implica la necesidad de llevar a cabo más codificaciones NewEdge sobre la estrategia `LRU`. Cómo se verá en las tasas de compresión obtenidas para las técnicas planteadas, la limitación del tamaño de ν a valores de $\alpha \in [2^9, 2^{10}]$ permiten obtener un *trade-off* espacio/tiempo muy competitivo.

Sin embargo, la efectividad de la técnica depende de un último parámetro: la asignación de identificadores a las aristas. Esta propiedad depende tanto del valor α como de la estrategia de ordenación utilizada. La figura 5.11 presenta los resultados obtenidos para las mismas



	LFU	LRU		LFU	LRU		LFU	LRU		LFU	LRU
1	27,36 %	19,06 %	5	3,21 %	4,31 %	9	1,68 %	2,29 %	13	0,94 %	1,22 %
2	14,05 %	12,28 %	6	3,15 %	2,87 %	10	1,48 %	2,19 %	14	0,83 %	1,12 %
3	6,32 %	5,47 %	7	2,48 %	2,82 %	11	1,17 %	1,72 %	15	0,78 %	1,03 %
4	4,32 %	4,44 %	8	2,33 %	2,36 %	12	1,02 %	1,35 %	16	0,69 %	0,95 %



	LFU	LRU		LFU	LRU		LFU	LRU		LFU	LRU
1	26,16 %	18,39 %	5	3,17 %	3,65 %	9	1,66 %	1,81 %	13	1,06 %	1,52 %
2	13,98 %	12,93 %	6	3,02 %	3,50 %	10	1,47 %	1,80 %	14	0,92 %	1,41 %
3	5,92 %	4,42 %	7	2,41 %	2,85 %	11	1,35 %	1,62 %	15	0,82 %	1,25 %
4	4,06 %	3,89 %	8	1,97 %	2,19 %	12	1,23 %	1,55 %	16	0,71 %	0,88 %

Figura 5.11: Distribución de los símbolos más y menos frecuentes en el flujo de codificaciones FOLLOW obtenido para las colecciones `WSJ010` (superior) y `FT94` (inferior) utilizando $\alpha = 2^{10}$

colecciones del caso anterior (`WSJ010` y `FT94`) al utilizar $\alpha = 2^{10}$. Las gráficas de la izquierda presentan la distribución de frecuencias de los 32 valores más utilizados mientras que las de la derecha representan el rango de valores $[33, 1024]$ con los que se representan las aristas menos significativas. Además, para cada colección, se muestra una tabla resumen con el porcentaje de utilización de los 16 símbolos más frecuentes.

La tendencia en ambas gráficas es similar, lo cual favorece a la estrategia de ordenación basada en frecuencia que requiere un menor número de codificaciones NEWEDGE. En primer lugar nos centraremos en los símbolos más frecuentes. El primer símbolo posee un porcentaje de aparición hasta 8 puntos mayor en `LFU`. Esto favorece la asignación de la palabra de código más corta para un mayor número de ocurrencias en el flujo principal. Las diferencias se acortan para el segundo y tercer símbolo, siendo a partir del cuarto/quinto cuando las tasas de aparición son mayores para `LRU`. Sin embargo, la gran diferencia existente en el símbolo más frecuente hace más sesgada la distribución de valores del flujo principal obtenido por `LFU` respecto a `LRU` que se

caracteriza por una distribución más equitativa. Haciendo un análisis global de los 16 símbolos más frecuentes, obtenemos que `LFU` cubre, respectivamente para `WSJ010` y `FT94`, el $71,83\,\%$ y $69,90\,\%$ de los símbolos totales mientras que `LRU` obtiene el $65,47\,\%$ y el $63,66\,\%$ en los mismos casos. Si nos fijamos en los símbolos menos frecuentes (gráficas derechas) podemos observar como el porcentaje de aparición de los símbolos `LRU` es igual o superior a los `LFU` que tiende a estar un "escalón" por debajo en ambos casos.

Además, se ha encontrado otro dato interesante en las distribuciones de frecuencias de los símbolos. La variación del valor de α tiene una influencia poco significativa en los valores más frecuentes. Esto supone que, al aumentar el tamaño de α, el número absoluto de ocurrencias de los valores más frecuentes apenas va a sufrir una ligera modificación. Por lo tanto, el beneficio obtenido en compresión al aumentar el valor de α está directamente ligado con el menor número de aristas codificadas. Esta propiedad favorece, desde dos perspectivas diferentes, cada una de las estrategias consideradas para la gestión de ν. Por un lado, utilizar vocabularios de tamaño limitado permite controlar la longitud máxima de las palabras de código utilizadas para codificar la función Follow. El precio pagado es el mayor número de funciones NewEdge utilizadas para codificar las aristas que entran y salen del modelo al limitar el grado de salida de los nodos. Por otro lado, utilizar el modelo completo permite minimizar el número de funciones NewEdge necesarias para la codificación del grafo aunque esto supone la necesidad de un alfabeto de entrada, a la función Follow, compuesto por un mayor número de elementos y, por tanto, una longitud media mayor para las palabras de código utilizadas.

Codificación orientada a bit (`E-G`$_1$). A continuación se plantea un resumen de los resultados obtenidos, en el presente entorno experimental, al fijar una codificación orientada a bit. Esta decisión conlleva establecer una gestión de dinámica de los símbolos de escape utilizados para la codificación del modelo. Tanto el flujo principal de enteros como el secundario, que representa la función función NewEdge, se codifican sobre un código de `Huffman` canónico. Para la implementación de este código se ha considerado la propuesta de Andrew Turpin[3].

La experimentación se lleva a cabo sobre una configuración heterogénea de colecciones de texto. Por un lado consideramos diferentes tamaños (aproximadamente 1, 10 y 100 MB.) para las colecciones `AP`, `WSJ` y `ZIFF` con el fin de analizar la evolución de los resultados ante un tamaño creciente del texto. Además, al seleccionar colecciones diferentes se consigue aislar los resultados obtenidos de las propiedades estilísticas particulares de cada colección. Desde esta misma perspectiva consideramos las colecciones `FTs` y `CR` con tamaños variables comprendidos entre 10 y 200 MB. Las tablas 5.4 y 5.5 resumen los resultados obtenidos en esta experimentación: la tabla 5.4 muestra los resultados obtenidos limitando el grado de salida máximo de los nodos y utilizando una ordenación/reemplazo `LFU` (izquierda) y `LRU` (derecha) mientras que la tabla 5.5, por su parte, presenta las ratios de compresión obtenidas con las variantes `estática` (izquierda) y `dinámica` (derecha) para las ordenación de los vocabularios asociados a los nodos representantes de palabras vacías. Finalmente se analiza el efecto de la elección de α considerando valores 2^3, 2^6, 2^9, 2^{10}, ∞. La columna $\alpha = \infty$ en `LFU` aparece sombreada ya que, cómo se explicaba anteriormente, este conjunto de valores es común para las variantes `LFU`, `estática` y `dinámica`.

Estos resultados respaldan varias conclusiones generales sobre la técnica `E-G`$_1$. Por un lado, plantea una opción poco competitiva para tamaños de texto pequeños debido a la sobrecarga inherente a la codificación del modelo. Éste es un resultado esperado de acuerdo a la Ley de Heaps (§3.1) dado que el tamaño del vocabulario para textos pequeños representa una fracción importante de su tamaño total. Por otro lado, la influencia del valor α es similar para todos los casos. Las ratios de compresión mejoran con valores crecientes de α alcanzando los mejores resultados para $\alpha = \infty$. Sin embargo, la eficiencia de la técnica se degrada de forma significativa

[3] http://www.cs.mu.oz.au/∼alistair/mr_coder/

Texto	LFU					LRU				
	2^3	2^6	2^9	2^{10}	∞	2^3	2^6	2^9	2^{10}	∞
AP001	30,03 %	28,16 %	27,62 %	27,61 %	27,66 %	30,93 %	31,29 %	31,35 %	30,94 %	29,63 %
AP010	27,29 %	24,79 %	23,82 %	23,71 %	23,62 %	27,99 %	27,88 %	28,05 %	28,02 %	25,69 %
AP100	26,64 %	23,62 %	22,26 %	21,99 %	21,71 %	26,64 %	26,83 %	27,07 %	26,99 %	23,70 %
WSJ001	29,46 %	27,54 %	27,13 %	27,12 %	27,11 %	30,26 %	30,54 %	30,59 %	30,29 %	29,02 %
WSJ010	27,09 %	24,45 %	23,50 %	23,44 %	23,44 %	27,83 %	27,86 %	27,86 %	27,80 %	25,71 %
WSJ100	26,15 %	23,16 %	21,88 %	21,61 %	21,42 %	27,02 %	26,87 %	26,99 %	26,83 %	23,68 %
ZIFF001	25,78 %	24,25 %	23,92 %	23,85 %	23,91 %	26,97 %	26,75 %	26,51 %	25,99 %	25,33 %
ZIFF010	25,35 %	23,64 %	22,77 %	22,75 %	22,79 %	26,43 %	26,68 %	26,50 %	26,48 %	24,42 %
ZIFF100	24,96 %	22,90 %	21,81 %	21,61 %	21,45 %	25,98 %	26,47 %	26,52 %	26,34 %	23,15 %
FT91	26,52 %	24,47 %	23,46 %	23,42 %	23,26 %	27,58 %	27,79 %	27,93 %	27,94 %	25,53 %
FT92	26,45 %	23,25 %	21,81 %	21,54 %	21,32 %	26,82 %	26,84 %	27,32 %	27,13 %	23,55 %
FT93	25,45 %	22,32 %	21,06 %	20,73 %	20,45 %	26,20 %	25,96 %	26,33 %	26,21 %	22,60 %
FT94	25,37 %	22,21 %	20,89 %	20,62 %	20,40 %	26,13 %	25,93 %	26,29 %	26,15 %	22,56 %
CR	24,39 %	21,62 %	20,52 %	20,30 %	20,15 %	24,61 %	24,50 %	24,50 %	24,55 %	21,75 %

Tabla 5.4: Tasas de compresión obtenidas con $\texttt{E-G}_1$ sobre las ordenación/reemplazo LFU y LRU.

Texto	Estática				Dinámica			
	2^3	2^6	2^9	2^{10}	2^3	2^6	2^9	2^{10}
AP001	29,57 %	28,47 %	27,91 %	27,77 %	28,96 %	28,14 %	27,80 %	27,71 %
AP010	26,15 %	24,94 %	24,20 %	24,02 %	25,47 %	24,46 %	23,93 %	23,82 %
AP100	24,84 %	23,59 %	22,65 %	22,41 %	24,07 %	22,86 %	22,21 %	22,05 %
WSJ001	29,02 %	27,92 %	27,36 %	27,24 %	28,48 %	27,56 %	27,22 %	27,16 %
WSJ010	25,86 %	24,71 %	23,92 %	23,77 %	25,28 %	24,20 %	23,68 %	23,58 %
WSJ100	24,43 %	23,24 %	22,26 %	22,02 %	23,72 %	22,50 %	21,83 %	21,68 %
ZIFF001	25,85 %	24,70 %	24,16 %	24,00 %	25,21 %	24,39 %	24,03 %	23,95 %
ZIFF010	25,38 %	24,13 %	23,35 %	23,16 %	24,82 %	23,68 %	23,10 %	22,97 %
ZIFF100	24,55 %	23,26 %	22,33 %	22,09 %	23,89 %	22,60 %	21,93 %	21,76 %
FT91	26,12 %	24,89 %	23,98 %	23,75 %	25,52 %	24,39 %	23,71 %	23,56 %
FT92	24,38 %	23,23 %	22,22 %	21,98 %	23,73 %	22,46 %	21,75 %	21,61 %
FT93	23,53 %	22,33 %	21,37 %	21,13 %	22,88 %	21,56 %	20,90 %	20,75 %
FT94	23,48 %	22,23 %	21,27 %	21,03 %	22,76 %	21,51 %	20,83 %	20,69 %
CR	22,85 %	21,75 %	20,84 %	20,65 %	22,19 %	21,09 %	20,49 %	20,37 %

Tabla 5.5: Tasas de compresión obtenidas con $\texttt{E-G}_1$ sobre las variantes estática y dinámica.

para $\alpha = \infty$ (ver tablas 5.6 y 5.7). El *trade-off* espacio/tiempo más competitivo se obtiene para $\alpha \in [2^9, 2^{10}]$: las ratios de compresión se aproximan notablemente al óptimo, obtenido para $\alpha = \infty$, con unos tiempos de compresión/descompresión hasta 12 veces mejor para colecciones de gran tamaño.

A continuación se analizan los resultados obtenidos por cada una de las estrategias de organización de ν. Los resultados obtenidos por LFU mejoran notablemente a LRU. Sin embargo, cada una de las variantes presenta una evolución diferente para los valores de α considerados. Por un lado, la efectividad obtenida por $\texttt{E-G}_1$ sobre LFU mejora con el incremento de α dado que la organización de las aristas más significativas apenas se ve alterada al aumentar este valor y, sin embargo, se reduce la cantidad de funciones NewEdge codificadas. Al ampliar la capacidad de ν se consigue un mayor equilibrio entre la cantidad de funciones NewEdge y Follow codificadas, alcanzando una representación más compacta ya que la función Follow tiende a necesitar un menor número de bits que NewEdge para su codificación.

Por su parte, $\texttt{E-G}_1$ sobre LRU muestra un comportamiento más estable para valores crecientes de α. Esto viene motivado por la ordenación más equitativa obtenida bajo el criterio de recencia. Por ejemplo, la diferencia entre LFU y LRU es superior a 5 puntos porcentuales para colecciones de gran tamaño ($100-200$ MB.) considerando $\alpha = 2^{10}$. Por lo tanto se concluye que LRU muestra un comportamiento poco competitivo para el problema actual y se descarta como solución práctica.

Las variantes estática y dinámica aprovechan la organización de ν obtenida por la estrategia basada en frecuencia. Como puede verse en la tabla 5.5, el comportamiento de ambas variantes muestra una mejoría en las ratios de compresión obtenidas para valores crecientes de α.

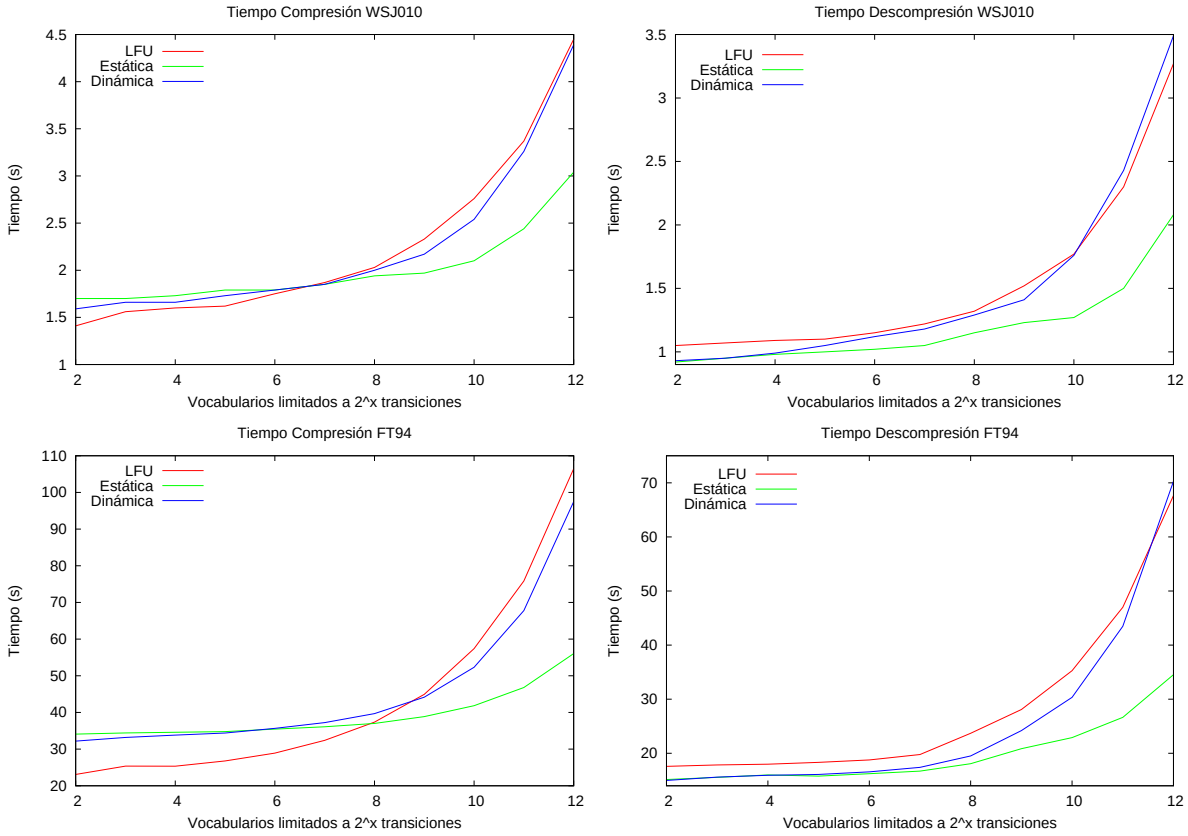


Figura 5.12: Tiempos de compresión/descompresión para `WSJ010` (superior) y `FT94` (inferior).

Texto	Variante	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	∞
WSJ010	LFU	1,41	1,56	1,60	1,62	1,75	1,87	2,03	2,33	2,76	3,37	4,45	
	Estática	1,70	1,70	1,73	1,79	1,79	1,85	1,94	1,97	2,10	2,44	3,04	6,32
	Dinámica	1,59	1,66	1,66	1,73	1,79	1,85	2,00	2,17	2,54	3,26	4,39	
FT94	LFU	23,08	25,36	25,34	26,78	28,91	32,38	37,37	44,94	57,42	75,84	106,46	
	Estática	34,10	34,42	34,60	34,77	35,45	36,08	37,00	38,87	41,84	46,79	56,04	487,00
	Dinámica	32,18	33,18	33,82	34,39	35,65	37,23	39,69	44,16	52,31	67,77	97,47	

Tabla 5.6: Tiempos de compresión obtenidos con $\texttt{E-G}_1$ para las variantes de implementación.

Texto	Variante	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	∞
WSJ010	LFU	1,05	1,07	1,09	1,10	1,15	1,22	1,32	1,52	1,77	2,30	3,27	
	Estática	0,92	0,95	0,98	1,00	1,02	1,05	1,15	1,23	1,27	1,50	2,08	2,876
	Dinámica	0,93	0,95	0,99	1,05	1,12	1,18	1,29	1,41	1,76	2,43	3,49	
FT94	LFU	17,59	17,85	17,99	18,33	18,78	19,77	23,71	28,09	35,27	47,00	67,63	
	Estática	15,16	15,56	16,00	15,78	16,23	16,71	18,08	20,85	22,91	26,64	34,54	442,15
	Dinámica	14,98	15,60	15,94	16,10	16,58	17,41	19,52	24,19	30,34	43,49	70,29	

Tabla 5.7: Tiempos de descompresión obtenidos con $\texttt{E-G}_1$ para las variantes de implementación.

La variante `dinámica` obtiene unos mejores resultados aprovechando la reordenación dinámica de las α primeras posiciones de ν de los nodos que representan palabras vacías. Sin embargo, este beneficio en espacio (respecto a la variante `estática`) tiene un coste en tiempo que empieza a ser considerable a partir de $\alpha = 2^{10}$ como puede observarse en la tendencia mostrada en la figura 5.12. El *trade-off* obtenido por las variantes `estática` y `dinámica` se aproxima progresivamente, para valores crecientes de α, hasta igualarse en $\alpha = \infty$ en el que ambas confluyen, al igual que la estrategia `LFU` previamente analizada.

Desde una perspectiva global, la utilización de una versión del modelo limitada a α aristas

por nodo es ligeramente más efectiva que la variante `dinámica` para los valores de α en los que se alcanza el mejor *trade-off* espacio/tiempo. Sin embargo, para valores de α menores, la variante `dinámica` obtiene unas mejores tasas de compresión. Por su parte, la variante `estática` mantiene un *trade-off* más uniforme gracias a los menores tiempos de cómputo obtenidos al no lleva a cabo una reorganización dinámica de ν. Nótese, sin embargo, que en términos prácticos la estrategia `LFU` plantea un consumo de memoria menor que los anteriores dado que mantiene únicamente una versión reducida del grafo.

La figura 5.12 muestra la evolución de los tiempos se compresión/descompresión de las colecciones `WSJ010` y `FT94` para cada una de las estrategias propuestas (se omite la representación de `LRU` previamente descartada). Las tablas 5.6 y 5.7 muestran los tiempos utilizados para representar las gráficas de la figura 5.12.

Como puede observarse en dicha figura, la estrategia `LFU` presenta los mejores tiempos de cómputo para valores pequeños de α dado que maneja un menor número de aristas al limitar el grado de salida máximo de los nodos. Por su parte, las variantes `estática` y `dinámica` presentan un comportamiento similar hasta $\alpha = 2^7$, debido a que la ordenación basada en frecuencia es capaz de mantener en las primeras posiciones las transiciones más significativas, de tal forma que éstas apenas se reordenan en la estrategia `dinámica`, obteniendo con ello una eficiencia similar a la obtenida por la `estática`. Sin embargo, a partir de este valor, se mantienen en el modelo transiciones menos significativas que sí requieren una mayor reordenación, degradando la eficiencia de la estrategia `dinámica` como puede observarse para valores de α desde 2^7. A partir de $\alpha = 2^7$, los tiempos de cómputo requeridos por la estrategia `estática` se desmarcan positivamente de los obtenidos por las otras dos. En el intervalo considerado competitivo, $\alpha \in [2^9, 2^{10}]$, dicha estrategia es un 10 %-31 % más rápida para `WSJ010` y un 13 %-37 % para `FT94`. Finalmente, se puede observar como la estrategia `dinámica` es ligeramente mejor que `LFU` a partir de $\alpha = 2^{10}$ (hasta un 7 % para `WSJ010` y 11 % para `FT94`), aunque como se observa en la tabla anterior, `LFU` es ligeramente más efectiva. El proceso de descompresión muestra una tendencia similar para las estrategias `estática` y `dinámica`. Sin embargo, la estrategia `LFU` es siempre más lenta que las anteriores para valores pequeños de α. Con el incremento de α, las estrategias `LFU` y `dinámica` se acercan, progresivamente, hasta que la segunda llega a superar ligeramente a la primera y, por tanto, necesitando unos tiempos de descompresión mayores.

Codificación orientada a byte (`E-G`$_1$`+X`). La siguiente parte del estudio muestra un análisis, equivalente al anterior, para la combinación de modelado `E-G`$_1$ y una codificación orientada a byte. Para este propósito se elige el código denso `ETDC` (ver §3.3.3). La propuesta actual representa de forma estática los símbolos de escape NewVertex y NewEdge mediante los códigos `ETDC` `<10000000>` y `<10000001>` (ver §5.3.5). La implementación del esquema de codificación adapta fuentes públicamente disponibles[4].

En primer lugar se lleva a cabo un análisis de los bytes que componen el flujo de codificación principal. Este estudio se basa en la experiencia previa presentada por Fariña *et al.* [FNP08] y sigue las mismas pautas que el planteado para `WCIM` en el capítulo anterior. Por lo tanto, nos centraremos en el análisis de las entropías de orden k (H_k) tanto del texto original (T) como de los flujos obtenidos de acuerdo a las diferentes estrategias de organización de ν. Estos valores muestran propiedades interesantes de las técnicas `E-G`$_1$`+X` analizadas. La tabla 5.8 muestra las entropías de orden k obtenidas para la colección `FT94`. Los valores H_k se calculan fijando $\alpha = 2^{10}$ para todos los casos.

La longitud media de una palabra en inglés es, aproximadamente, 5 bytes mientras que su varianza es relativamente alta. Sin embargo, un texto procesado sobre el modelo `E-G`$_1$`+X` consigue una longitud media de palabra inferior a 2 bytes. Esto supone, a su vez, una varianza menor

[4] http://vios.dc.fi.udc.es/codes/download.html

k	Plano		LFU		LRU		Estática		Dinámica	
	H_k	Contextos	H_k	Contextos	H_k	Contextos	H_k	Contextos	H_k	Contextos
0	5,0006	1	5,9847	1	6,2287	1	6,1737	1	6,1448	1
1	3,6388	88	5,2865	256	5,5114	256	5,6269	256	5,6.090	256
2	2,7840	5.113	4,9511	36.089	5,1823	36.089	5,2611	47.977	5,2676	48.148
3	2,1166	74.498	4,2649	1.910.536	4,4068	2.290.188	4,2163	2.700.582	4,2735	2.701.968
4	1,6963	524.699	2,5613	14.488.638	2,3755	18.773.511	2,0392	18.981.910	2,1057	19.105.631
5	1,4611	2.072.515	0,9849	35.986.426	0,7629	43.229.600	0,6389	39.211.801	0,6506	39.893.004
6	1,3078	5.420.628	0,3250	48.770.657	0,2556	53.962.925	0,2360	47.445.669	0,2275	48.242.105
7	1,1735	10.943.311	0,1602	52.897.341	0,1502	56.648.804	0,1464	49.776.943	0,1437	50.315.202
8	1,0417	18.788.845	0,1200	54.177.143	0,1166	57.598.356	0,1153	50.714.667	0,1190	51.078.926
9	0,9120	28.878.013	0,1024	54.810.237	0,1036	58.202.040	0,1001	51.284.151	0,1048	51.570.565
10	0,7893	40.693.718	0,0937	55.267.065	0,0932	58.686.030	0,0936	51.704.161	0,0972	51.957.147
20	0,1285	141.687.067	0,0382	58.389.187	0,0277	61.957.596	0,0419	54.593.717	0,0393	54.956.336
30	0,0392	164.993.841	0,0209	60.565.790	0,0093	63.579.049	0,0215	56.681.528	0,0186	57.043.607
40	0,0242	171.102.511	0,0062	61.836.803	0,0034	64.088.913	0,0096	58.016.197	0,0067	58.169.194
50	0,0195	174.979.550	0,0021	62.455.564	0,0017	64.329.941	0,0037	58.840.491	0,0021	58.758.035

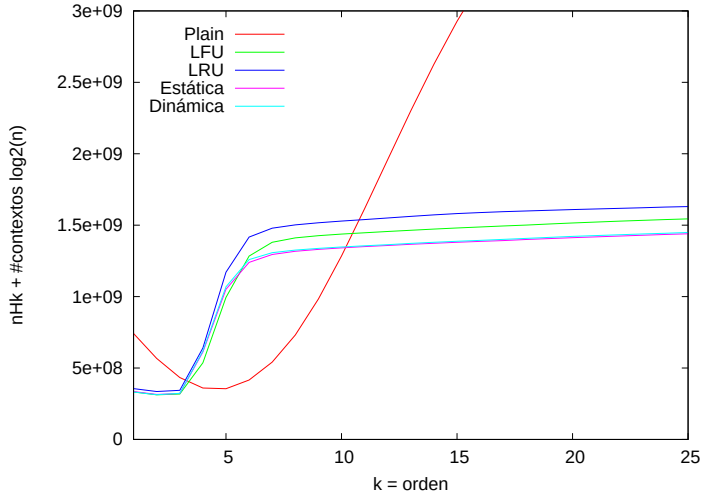


Tabla 5.8: Estudio de las entropías de orden k para la colección `FT94` y valores $nHk + \#contextos\ log_2(n)$ obtenidos para el mismo.

que para el caso anterior dado que difícilmente se utilizan códigos de 3 bytes (se necesitaría la existencia de un nodo con 2.113.664 aristas). Esto supone que, como se concluye en el trabajo previamente referenciado, un modelo de orden superior es capaz de identificar transiciones entre palabras consecutivas con un valor menor de k o capturar correlaciones mayores para un valor de k determinado.

La comparación actual considera una penalización de $log_2(n)$ bits [FNP08] por cada contexto utilizado. La figura 5.8 muestra esta comparativa. Como puede observarse, `LFU` y la variante `dinámica` alcanzan el valor mínimo en la gráfica (ligeramente mejor que las otras propuestas). Por su parte, el modelado del texto plano requiere un mayor valor de k para alcanzar su mínimo. En lo que respecta a cada una de las variantes, puede observarse como `LFU` es ligeramente inferior a las otras hasta $k = 6$. A partir de este valor se comporta peor que el resto de estrategias (a excepción de `LRU`) que definen una evolución similar para valores crecientes de k. Este análisis nos permite concluir que `E-G`$_1$`+X`, sobre ordenación/reemplazo `LFU`, es ligeramente más efectiva que el resto de variantes. Al igual que en el caso anterior, `LRU` presenta un peor rendimiento (ver [AMPdlF07]), por lo que se descarta su viabilidad dado que, en ningún caso, mejora los resultados obtenidos por el resto de variantes.

Las tablas 5.9, 5.10, 5.11 y 5.12 muestran un resumen de las tasas de compresión obtenidas para el modelo `E-G`$_1$ actual sobre una codificación `ETDC` orientada a byte. Al igual que para el caso

Texto	LFU			Estática			Dinámica			∞	ETDC
	2^6	2^9	2^{10}	2^6	2^9	2^{10}	2^6	2^9	2^{10}		
AP001	51,38 %	49,36 %	49,03 %	49,57 %	49,12 %	48,91 %	49,57 %	48,95 %	48,84 %	48,83 %	**44,21 %**
AP010	42,06 %	38,93 %	38,27 %	38,40 %	37,76 %	37,50 %	38,40 %	37,40 %	37,28 %	37,23 %	**36,15 %**
AP100	38,34 %	34,33 %	33,40 %	32,94 %	32,10 %	31,79 %	32,94 %	31,65 %	31,51 %	**31,37 %**	33,34 %
WSJ001	51,09 %	49,15 %	48,88 %	49,34 %	48,91 %	48,77 %	49,34 %	48,77 %	48,70 %	48,69 %	**43,98 %**
WSJ010	42,05 %	38,95 %	38,39 %	38,53 %	37,88 %	37,69 %	38,53 %	37,61 %	37,52 %	37,50 %	**36,12 %**
WSJ100	37,93 %	33,83 %	33,05 %	32,79 %	31,91 %	31,65 %	32,79 %	31,52 %	31,40 %	**31,33 %**	33,23 %
ZIFF001	45,76 %	43,98 %	43,79 %	44,44 %	43,93 %	43,77 %	44,44 %	43,80 %	43,70 %	43,69 %	**40,80 %**
ZIFF010	41,12 %	38,19 %	37,58 %	37,89 %	37,23 %	36,98 %	37,89 %	36,93 %	36,78 %	36,73 %	**36,09 %**
ZIFF100	38,23 %	34,48 %	33,72 %	33,51 %	32,66 %	32,37 %	33,51 %	32,28 %	32,11 %	**32,01 %**	33,97 %
FT91	41,80 %	38,66 %	38,07 %	38,50 %	37,72 %	37,45 %	38,50 %	37,41 %	37,27 %	37,07 %	**35,53 %**
FT92	37,70 %	33,49 %	32,71 %	32,37 %	31,41 %	31,13 %	32,37 %	30,98 %	30,87 %	**30,79 %**	32,82 %
FT93	36,81 %	32,92 %	32,15 %	31,76 %	30,91 %	30,62 %	31,76 %	30,49 %	30,36 %	**30,26 %**	32,87 %
FT94	36,74 %	32,83 %	32,09 %	31,71 %	30,78 %	30,51 %	31,71 %	30,41 %	30,29 %	**30,20 %**	32,83 %
CR	35,76 %	32,40 %	31,76 %	31,68 %	30,89 %	30,67 %	31,68 %	30,53 %	30,43 %	**30,38 %**	31,94 %

Tabla 5.9: Tasas de compresión "en plano" obtenidas con E-G_1

Texto	LFU			Estática			Dinámica			∞	ppmdi
	2^6	2^9	2^{10}	2^6	2^9	2^{10}	2^6	2^9	2^{10}		
AP001	**26,37 %**	26,44 %	26,46 %	26,67 %	26,58 %	26,56 %	26,63 %	26,58 %	26,57 %	26,53 %	26,43 %
AP010	23,30 %	23,28 %	**23,26 %**	23,70 %	23,46 %	23,38 %	23,61 %	23,43 %	23,40 %	23,35 %	25,59 %
AP100	21,87 %	21,70 %	**21,65 %**	22,30 %	21,88 %	21,78 %	22,10 %	21,81 %	21,77 %	21,69 %	25,64 %
WSJ001	25,70 %	25,87 %	25,93 %	26,18 %	26,05 %	26,03 %	26,15 %	26,07 %	26,03 %	26,00 %	**25,61 %**
WSJ010	**22,99 %**	**22,99 %**	23,02 %	23,57 %	23,32 %	23,24 %	23,48 %	23,28 %	23,22 %	23,14 %	25,42 %
WSJ100	21,56 %	**21,35 %**	**21,35 %**	22,23 %	21,72 %	21,62 %	21,96 %	21,61 %	21,55 %	21,40 %	25,40 %
ZIFF001	22,62 %	22,83 %	22,88 %	22,90 %	22,86 %	22,91 %	22,90 %	22,93 %	22,94 %	22,92 %	**21,08 %**
ZIFF010	**22,13 %**	22,32 %	22,40 %	22,77 %	22,54 %	22,45 %	22,68 %	22,52 %	22,52 %	22,53 %	22,97 %
ZIFF100	**21,19 %**	21,25 %	21,31 %	22,03 %	21,60 %	21,49 %	21,80 %	21,52 %	21,49 %	21,49 %	23,17 %
FT91	22,88 %	**22,74 %**	**22,74 %**	23,72 %	23,23 %	23,07 %	23,51 %	23,11 %	23,02 %	22,83 %	25,30 %
FT92	21,57 %	21,20 %	21,16 %	22,16 %	21,57 %	21,44 %	21,84 %	21,41 %	21,34 %	**21,09 %**	25,34 %
FT93	19,98 %	19,69 %	**19,64 %**	20,44 %	19,92 %	19,79 %	20,16 %	19,82 %	19,76 %	**19,64 %**	23,55 %
FT94	19,95 %	19,60 %	**19,55 %**	20,41 %	19,85 %	19,71 %	20,16 %	19,75 %	19,67 %	19,56 %	23,55 %
CR	19,74 %	**19,68 %**	19,70 %	20,15 %	19,82 %	19,72 %	20,01 %	19,80 %	19,81 %	19.88 %	22,42 %

Tabla 5.10: Tasas de compresión obtenidas con E-G_1+ppmdi.

anterior, se considera una colección heterogénea de textos y el mismo conjunto de valores de α, del que se descarta $\alpha = 2^3$ dado que considerar cómo palabras vacías aquellas que se relacionen con más de 2^3 palabras diferentes no representa la realidad de un texto en lenguaje natural. Las mejores tasas de compresión obtenidas, para cada caso, aparecen resaltadas. La tabla 5.9 muestra los resultados "en plano" obtenidos para el modelado E-G_1 y su codificación sobre ETDC, la tabla 5.10 plantea las ratios obtenidas por E-G_1+ppmdi, la tabla 5.11 las de E-G_1+bzip2 y, finalmente, la tabla 5.12 muestra los resultados obtenidos con E-G_1+p7zip. Las propiedades de los compresores ppmdi, bzip2 y p7zip están completamente detalladas en el apéndice A.

La última columna de la tabla 5.9 muestra los resultados obtenidos al utilizar el propio código ETDC sobre un modelo de palabras de orden 0 (ver §3.3.3). Esta propuesta mejora siempre los resultados obtenidos por las diferentes variantes de E-G_1 para los ficheros de pequeño tamaño. Cómo se comentaba para la codificación orientada a bit, esto se debe a la necesidad de codificar el grafo utilizado. Sin embargo, los resultados de E-G_1 son siempre mejores para tamaños medianos y grandes aprovechando el modelado de orden 1 en el que se basa. Entre las variantes consideradas, puede observarse como LFU obtiene las peores tasas de compresión debido a la necesidad de codificar un mayor número de funciones NewEdge. A pesar de ello, el estudio anterior sobre las entropías de orden k muestra que ésta representa la variante más efectiva debido, principalmente, a la utilización de un alfabeto de entrada de menor tamaño que permite obtener una distribución de valores más sesgada. Por su parte, tanto la variante estática como la dinámica muestran resultados similares, ligeramente favorables a la segunda. Finalmente, puede observarse como, en término globales, las mejores ratios se obtienen para $\alpha = \infty$.

Texto	LFU			Estática			Dinámica			∞	bzip2
	2^6	2^9	2^{10}	2^6	2^9	2^{10}	2^6	2^9	2^{10}		
AP001	**28,22 %**	28,32 %	28,35 %	28,61 %	28,50 %	28,47 %	28,60 %	28,48 %	28,48 %	28,46 %	28,31 %
AP010	24,92 %	24,89 %	**24,87 %**	25,26 %	24,95 %	**24,87 %**	25,19 %	24,92 %	24,93 %	24,94 %	27,20 %
AP100	23,39 %	23,21 %	23,18 %	23,83 %	23,30 %	23,18 %	23,65 %	23,18 %	**23,17 %**	**23,17 %**	27,23 %
WSJ001	27,52 %	27,68 %	27,76 %	28,08 %	27,91 %	27,87 %	28,07 %	27,91 %	27,88 %	27,87 %	**27,45 %**
WSJ010	**24,56 %**	24,59 %	24,59 %	25,06 %	24,76 %	24,68 %	25,01 %	24,74 %	24,70 %	24,68 %	26,92 %
WSJ100	23,07 %	**22,88 %**	22,90 %	23,68 %	23,11 %	23,01 %	23,48 %	22,98 %	22,97 %	22,90 %	26,86 %
ZIFF001	24,52 %	24,74 %	24,77 %	24,84 %	24,78 %	24,83 %	24,86 %	24,86 %	24,87 %	24,87 %	**23,29 %**
ZIFF010	**23,92 %**	24,07 %	24,18 %	24,48 %	24,22 %	24,13 %	24,43 %	24,19 %	24,21 %	24,29 %	25,00 %
ZIFF100	**22,94 %**	22,95 %	23,03 %	23,70 %	23,18 %	23,07 %	23,51 %	23,08 %	23,08 %	23,16 %	25,21 %
FT91	24,39 %	**24,24 %**	24,26 %	25,14 %	24,63 %	24,49 %	24,98 %	24,51 %	24,44 %	24,32 %	27,06 %
FT92	23,09 %	22,69 %	22,66 %	23,62 %	22,93 %	22,82 %	23,35 %	22,75 %	22,70 %	**22,53 %**	27,10 %
FT93	21,48 %	21,13 %	21,10 %	21,89 %	21,29 %	21,16 %	21,66 %	21,16 %	21,12 %	**21,06 %**	25,32 %
FT94	21,40 %	21,02 %	21,01 %	21,81 %	21,20 %	21,08 %	21,60 %	21,07 %	21,04 %	**20,97 %**	25,27 %
CR	21,20 %	21,10 %	21,14 %	21,56 %	21,15 %	**21,07 %**	21,45 %	21,14 %	21,17 %	21,29 %	24,14 %

Tabla 5.11: Tasas de compresión obtenidas con E-G_1+bzip2.

Texto	LFU			Estática			Dinámica			∞	p7zip
	2^6	2^9	2^{10}	2^6	2^9	2^{10}	2^6	2^9	2^{10}		
AP001	28,64 %	28,62 %	28,60 %	28,72 %	28,60 %	**28,58 %**	28,67 %	28,59 %	28,60 %	28,59 %	28,95 %
AP010	24,42 %	24,47 %	24,45 %	24,42 %	24,47 %	24,45 %	24,52 %	**24,41 %**	24,42 %	24,44 %	24,91 %
AP100	21,99 %	22,10 %	22,08 %	21,99 %	22,10 %	22,08 %	21,94 %	**21,90 %**	21,92 %	22,01 %	24,21 %
WSJ001	**28,03 %**	28,06 %	28,06 %	28,28 %	28,11 %	28,08 %	28,23 %	28,11 %	28,10 %	28,08 %	29,00 %
WSJ010	24,29 %	**24,24 %**	24,26 %	24,51 %	24,33 %	24,26 %	24,51 %	24,34 %	24,30 %	24,28 %	26,00 %
WSJ100	22,15 %	21,97 %	21,95 %	22,27 %	21,90 %	21,83 %	22,12 %	21,91 %	21,87 %	**21,79 %**	25,38 %
ZIFF001	**24,57 %**	24,68 %	24,71 %	24,67 %	24,62 %	24,68 %	24,66 %	24,69 %	24,71 %	24,71 %	25,20 %
ZIFF010	**23,32 %**	23,47 %	23,56 %	23,70 %	23,50 %	23,44 %	23,64 %	23,51 %	23,54 %	23,63 %	24,60 %
ZIFF100	21,77 %	21,82 %	21,88 %	22,13 %	21,77 %	**21,69 %**	21,96 %	21,77 %	21,77 %	21,85 %	24,45 %
FT91	24,15 %	23,97 %	23,93 %	24,62 %	24,20 %	24,07 %	24,51 %	24,14 %	24,07 %	**23,90 %**	26,11 %
FT92	22,09 %	21,73 %	21,64 %	22,11 %	21,60 %	21,49 %	21,83 %	21,52 %	21,45 %	**21,28 %**	25,55 %
FT93	20,54 %	20,26 %	20,18 %	20,43 %	19,98 %	19,87 %	20,19 %	19,97 %	19,92 %	**19,84 %**	23,82 %
FT94	20,42 %	20,14 %	20,06 %	20,32 %	19,86 %	19,76 %	20,14 %	19,87 %	19,82 %	**19,74 %**	23,75 %
CR	20,31 %	20,29 %	20,32 %	20,26 %	20,02 %	**19,96 %**	20,23 %	20,13 %	20,18 %	20,38 %	22,93 %

Tabla 5.12: Tasas de compresión obtenidas con E-G_1+p7zip.

La tabla 5.10 muestra los resultados de la técnica E-G_1+ppmdi y los compara con los obtenidos por la técnica original ppmdi (parametizada con la opción por defecto -l6). Los resultados obtenidos confirman las conclusiones extraídas del estudio de entropías: la variante LFU es la más efectiva de las tres. La diferencia respecto a la variante dinámica es poco significativa, aunque es interesante observar como LFU obtiene mejores resultados para $\alpha \in [2^9, 2^{10}]$ que al fijar $\alpha = \infty$ que representa el valor óptimo para la variante dinámica. En todos los casos, excepto para las colecciones de menor tamaño, E-G_1+ppmdi mejora los resultados obtenidos por ppmdi, obteniendo diferencias significativas de hasta 4 puntos porcentuales para las colecciones de mayor tamaño. En lo que respecta a la eficiencia, los procesos de compresión en E-G_1+ppmdi (ver tabla 5.13) requieren un tiempo de cómputo mayor que en ppmdi. El motivo principal de este coste es la necesidad de generar y actualizar el modelo E-G_1 y, posteriormente, recodificar su resultado con la técnica ppmdi. En la misma tabla puede observarse como la diferencia en tiempos es mayor para valores crecientes de α. E-G_1+ppmdi es capaz de mejorar, en algunos casos, los tiempos de descompresión obtenidos por ppmdi. En este proceso, E-G_1+ppmdi aprovecha la representación implícita del modelo en el texto codificado, de tal forma su reconstrucción se lleva a cabo de forma eficiente y su actualización es más rápida que en el proceso de compresión, de acuerdo al estudio de costes presentado en §5.3.4.

La tabla 5.11 muestra los resultados obtenidos por E-G_1+bzip2, cuya interpretación es similar a la anterior. Nótese que los tiempos de compresión/descompresión (tabla 5.14) son mejores que para el caso anterior debido a la mayor eficiencia de bzip2 respecto a ppmdi. Finalmente, la tabla 5.12 resume las tasas de compresión obtenidas al combinar el preprocesamiento E-G_1 con

		Compresión					Descompresión				
		2^6	2^9	2^{10}	∞	ppmdi	2^6	2^9	2^{10}	∞	unppmdi
WSJ010	LFU	2,28	3,87	4,31			1,88	3,07	3,36		
	Estática	2,39	3,74	3,90	8,10	1,99	1,90	2,84	2,98	6,93	2,04
	Dinámica	2,37	3,93	4,32			1,94	3,04	3,37		
		2^6	2^9	2^{10}	∞	ppmdi	2^6	2^9	2^{10}	∞	unppmdi
FT94	LFU	58,63	69,32	78,12			47,83	53,51	60,48		
	Estática	71,01	71,70	74,72	517,58	55,53	51,25	50,34	53,05	471,74	57,24
	Dinámica	70,78	76,41	84,66			48,89	54,17	60,99		

Tabla 5.13: Tiempos de compresión/descompresión obtenidos con E-G_1+ppmdi para la colecciones WSJ010/FT94 y comparativa con ppmdi/unppmdi.

		Compresión					Descompresión				
		2^6	2^9	2^{10}	∞	bzip2	2^6	2^9	2^{10}	∞	bunzip2
WSJ010	LFU	1,45	2,64	3,05			0,78	1,49	1,81		
	Estática	1,50	2,37	2,55	6,92	0,97	0,72	1,23	1,30	5,28	0,40
	Dinámica	1,49	2,67	3,06			0,75	1,38	1,71		
		2^6	2^9	2^{10}	∞	bzip2	2^6	2^9	2^{10}	∞	bunzip2
FT94	LFU	35,66	47,00	57,00			18,38	25,67	32,49		
	Estática	44,27	48,19	50,95	505,55	30,74	17,47	20,69	23,64	453,05	11,60
	Dinámica	43,78	53,39	60,88			18,04	25,12	25,98		

Tabla 5.14: Tiempos de compresión/descompresión obtenidos con E-G_1+bzip2 para la colecciones WSJ010/FT94 y comparativa con bzip2/bunzip2.

		Compresión					Descompresión				
		2^6	2^9	2^{10}	∞	p7zip	2^6	2^9	2^{10}	∞	p7zip -d
WSJ010	LFU	2,80	2,88	2,99			0,75	0,91	1,10		
	Estática	2,48	2,54	2,64	4,69	6,39	0,64	0,76	0,83	2,77	0,26
	Dinámica	2,49	2,68	2,87			0,68	0,88	1,05		
		2^6	2^9	2^{10}	∞	p7zip	2^6	2^9	2^{10}	∞	p7zip -d
FT94	LFU	91,29	102,32	107,68			16,41	23,83	30,70		
	Estática	87,42	89,77	92,75	541,22	246,46	15,74	18,68	21,07	442,44	4,77
	Dinámica	89,38	94,65	102,93			17,64	23,34	29,94		

Tabla 5.15: Tiempos de compresión/descompresión obtenidos con E-G_1+p7zip para la colecciones WSJ010/FT94 y comparativa con p7zip/p7zip -d.

la técnica de compresión p7zip. Los resultados se sitúan entre los obtenidos por E-G_1+bzip2 y los de E-G_1+ppmdi, aunque en este caso la mejora respecto al compresor original es menor para p7zip. Además, E-G_1+p7zip no es capaz de beneficiarse de la alta velocidad de descompresión que caracteriza a p7zip (ver tabla 5.15) y, a su vez, mantiene el alto coste temporal derivado de su proceso de compresión. Aún así, E-G_1+bzip2 mejora hasta 4 puntos porcentuales la efectividad de bzip2 mientras que E-G_1+p7zip obtiene una mejora de hasta $3,5$ puntos porcentuales respecto a p7zip.

A tenor de los resultados presentados en la sección actual podemos concluir que la técnica E-G_1 posee un *trade-off* espacio/tiempo muy competitivo gracias al conjunto de variantes consideradas tanto para su modelado como para su codificación. Las variantes orientadas a bit, sobre Huffman canónico, se caracterizan por procesos de compresión/descompresión más rápidos que E-G_1+X aunque con unas tasas de compresión ligeramente inferiores respecto a E-G_1+ppmdi. Sin embargo, las ratios obtenidas con codificación Huffman se pueden mejorar al aumentar el valor α que gobierna las variantes de organización de ν a cambio de reducir la eficiencia de sus procesos de cómputo. De la misma forma, al reducir α en E-G_1+X se puede ganar en eficiencia a costa de perder efectividad. De esta manera, la parametrización de E-G_1 facilita su adaptación a diferentes contextos de aplicación de acuerdo a las necesidades que éstos planteen, centrados en procesos más rápidos o esperando un espacio de almacenamiento menor.

Texto	k	Nodos $\|N\|$		Aristas $\|A\|$		Símbolos $\|T\|$		$\|A\|/\|N\|$	$\|T\|/\|A\|$
WSJ010	1	**53832**		**573453**		**2217389**		10,65	x
	2	56010	(4,05 %)	632487	(10,29 %)	1715053	(−22,65 %)	11,29	2,71
	3	56911	(5,72 %)	654161	(14,07 %)	1581567	(−28,67 %)	11,49	2,42
	4	57988	(7,72 %)	669717	(16,79 %)	1513789	(−31,73 %)	11,55	2,26
	5	58584	(8,83 %)	675832	(17,85 %)	1485636	(−33,00 %)	11,54	2,20
Texto	k	**Nodos** $\|N\|$		**Aristas** $\|A\|$		**Símbolos**		$\|A\|/\|N\|$	$\|T\|/\|A\|$
CR	1	**117718**		**1609786**		**10113132**		13,67	x
	2	122397	(3,97 %)	2000777	(24,29 %)	7531108	(−25,53 %)	16,35	3,76
	3	124779	(6,00 %)	2105504	(30,79 %)	6823371	(−32,53 %)	16,87	3,24
	4	127718	(8,49 %)	2179476	(35,39 %)	6415561	(−36,56 %)	17,06	2,94
	5	129472	(9,98 %)	2211030	(37,35 %)	6240648	(−38,29 %)	17,08	2,82
Texto	k	**Nodos** $\|N\|$		**Aristas** $\|A\|$		**Símbolos**		$\|A\|/\|N\|$	$\|T\|/\|A\|$
WSJ100	1	**149658**		**2856555**		**22161021**		19,09	x
	2	155640	(4,00 %)	3768466	(31,92 %)	16354985	(−26,20 %)	24,21	4,34
	3	158710	(6,05 %)	4043831	(41,56 %)	14731299	(−33,53 %)	25,48	3,64
	4	162593	(8,64 %)	4232085	(48,15 %)	13932129	(−37,13 %)	26,03	3,29
	5	164838	(10,14 %)	4306295	(50,75 %)	13606114	(−38,60 %)	26,12	3,16
Texto	k	**Nodos** $\|N\|$		**Aristas** $\|A\|$		**Símbolos**		$\|A\|/\|N\|$	$\|T\|/\|A\|$
FT94	1	**295018**		**4400311**		**43039675**		14,92	x
	2	305698	(3,62 %)	6070232	(37,95 %)	30326868	(−29,54 %)	19,86	5,00
	3	311281	(5,51 %)	6638444	(50,86 %)	26515515	(−38,39 %)	21,33	3,99
	4	318668	(8,02 %)	6921427	(57,29 %)	24641512	(−42,75 %)	21,72	3,56
	5	323304	(9,59 %)	7061446	(60,48 %)	23744377	(−44,83 %)	21,84	3,36

Tabla 5.16: Tamaños de los modelos Edge-Guided$_k$ para textos de diferentes tamaños.

5.5.3. Resultados experimentales para E-G$_k$

La extensión del alfabeto original de palabras supone una evolución en el tamaño del modelo utilizado en la técnica actual. Por un lado, $|\mathcal{N}|$ aumenta ante la necesidad de representar las nuevas secuencias de palabras consideradas en el alfabeto extendido. Sin embargo, éste no es el valor más determinante en el análisis del coste de representación del nuevo modelo. Al igual que en el caso base, $|\mathcal{A}|$ es el que determina la eficiencia espacio-temporal de la propuesta actual.

La tabla 5.16 muestra las configuraciones de nodos y aristas obtenidas al representar diferentes colecciones con el modelo E-G$_k$. La primera fila de cada subtabla muestra los datos obtenidos en el caso base (similares a los presentados en la tabla 5.2) que se utilizan como referencia para estimar la evolución del tamaño de la nueva propuesta. El algoritmo que identifica los q-gramas de palabras con los que se extiende el alfabeto original procede en diferentes *rounds*. En cada uno de ellos se forman nuevos q-gramas a partir de la configuración del alfabeto en el *round* justo anterior. En la presente tabla se muestran los resultados obtenidos sobre procesos de identificación de q-gramas compuestos por $2, 3, 4$ y 5 *rounds*. La columna *símbolos* contempla el número total de símbolos identificados de acuerdo al nuevo alfabeto. El porcentaje mostrado, junto a cada valor, la reducción de $|\mathcal{T}|$ al considerar el alfabeto extendido respecto al original de palabras utilizados en el caso base.

El valor $|\mathcal{N}|$ crece progresivamente con el valor de k, sin embargo es un crecimiento controlado entre el 4 % y el 10 % del tamaño del alfabeto original. Esto supone un ligero incremento en el espacio requerido para la representación de $|\mathcal{N}|$. Sin embargo, $|\mathcal{A}|$ presenta un crecimiento mayor. Aunque sigue siendo un incremento lineal con el número de aristas originalmente representadas, en términos prácticos, supone un aumento significativo del coste de almacenamiento. Se puede observar como el aumento de $|\mathcal{A}|$ está directamente ligado con $|\mathcal{N}|$, de tal forma que para las colecciones de mayor tamaño este incremento es más pronunciado (para $k = 5$, $|A|$ aumenta un 17,85 % para WSJ010 mientras que para FT94 este incremento es del 60,48 %). La relación $|\mathcal{A}|/|\mathcal{N}|$, que determina el tamaño del grafo requerido, mantiene una tendencia lineal aunque en este caso el valor m' tal que $|\mathcal{A}| = m'|\mathcal{N}|$ es superior al valor m determinado para el caso base. El tamaño del nuevo modelo es m'/m veces mayor que el del modelo original. Además,

se debe considerar que el compresor necesita tanto la gramática como la secuencia comprimida obtenidas en la generación de los q-gramas. Estas estructuras son similares a las obtenidas en el algoritmo `Re-Pair`. Esto supone un pequeño coste de $2\Delta|\mathcal{N}|$ para la gramática dado que ésta utiliza dos palabras de memoria por regla para representar los dos componentes que definen cada uno de los q-gramas añadidos al modelo.

Por su parte, la secuencia comprimida $\mathcal{S}$ requiere una palabra de memoria por cada símbolo en el texto. La última columna de la tabla ($|\mathcal{T}|/|\mathcal{A}|$) da una estimación del coste que supone almacenar $\mathcal{S}$; esto supone que, por ejemplo, el tamaño de la secuencia comprimida tiene un coste entre $2,82$ y $3,76$ veces el tamaño del modelo para el texto `CR`. Nótese que estas dos estructuras sólo se necesitan en el proceso de compresión, por lo que el coste del descompresor mantiene un orden de complejidad $O(2m'\sigma)$.

`E-G`$_k$ comparte las propiedades analizadas para el caso base, por lo que la elección de α tiene un importante impacto en la eficiencia y la efectividad de la técnica actual. Se debe considerar que para el caso actual, el número medio de aristas por nodo ($|\mathcal{A}|/|\mathcal{N}|$) se incrementa respecto al caso base aunque, en términos prácticos, el grado de salida de los nodos que representan palabras vacías se reduce. Esto supone un menor número de reemplazos para las mismas condiciones definidas en el caso anterior, lo que traslada ligeramente hacia abajo las curvas `LFU` y `LRU` mostradas en la figura 5.10. Por su parte, la recta *modelo completo* (en la misma imagen) se desplaza hacia arriba de acuerdo a la evolución de $|\mathcal{A}|$. Dicha evolución tiene una doble repercusión al limitar el tamaño máximo de ν. Por un lado, degrada la eficiencia temporal dado que los reemplazos se distribuyen de manera más uniforme entre un conjunto mayor de nodos, lo que implica un coste mayor que el soportado en el caso base. Por otro lado, al reducirse el grado de salida medio de los nodos que representan palabras vacías, la efectividad de la estrategia `LFU` mejora respecto a las estrategias `estática` y `dinámica` dado que se reduce el número de codificaciones NewEdge necesarias. Aún así, estas propiedades tienen un ligero impacto sobre la codificación de las transiciones más relevantes dado que éstas tienden a caracterizarse por una baja reordenación. Esto supone que, para tamaños grandes de α (a partir del valor 2^7 deducido anteriormente) las tasas de compresión para $\alpha = 2^x$ y $\alpha = 2^{x+1}$ tienden a ser más cercanas que en el caso base.

La eficiencia temporal asociada a la presente propuesta se degrada respecto a la original dado que precisa manejar un modelo más complejo que en el caso anterior. En primer lugar, la construcción del modelo es más costosa dado que precisa manejar la relación de jerarquía existente entre los q-gramas, utilizando para ello el mecanismo de *blending* propuesto. A su vez, se debe considerar que los q-gramas, con los que se extiende el alfabeto original, se forman a partir de transiciones muy frecuentes que tienden a formar secuencias de palabras vacías. Esto supone un incremento del número de nodos cuyo grado de salida es mayor que α y con ello el coste medio de las operaciones realizadas sobre sus vocabularios de transiciones.

A continuación se lleva a cabo un estudio experimental de la presente técnica `E-G`$_k$. El estudio de los resultados se divide de acuerdo a la codificación utilizada en cada caso. Sin embargo, para el caso actual, no se profundiza en las técnicas `E-G`$_k$`+X` dado que no mejoran las tasas de compresión obtenidas por las variantes orientadas a bit y, a su vez, requieren procesos de cómputo más intensivos en espacio y tiempo.

Codificación orientada a bit (`E-G`$_k$). La implementación de la propuesta actual añade el manejo de las relaciones de jerarquía y la gestión dinámica del código de escape NewContext a la base considerada en `E-G`$_1$.

Las tablas 5.17, 5.18 y 5.19 resumen los resultados obtenidos por la técnica actual. `E-G`$_k$ referencia la ejecución del algoritmo actual sobre un proceso de identificación de q-gramas en $k-1$ *rounds*. Se consideran tres textos de diferente tamaño en la colección `WSJ` para analizar

la evolución de las tasas de compresión con el tamaño del texto. A su vez se consideran las colecciones FTs y CR para disponer de un conjunto heterogéneo de textos. Se utilizan diferentes valores de α (2^6, 2^9, 2^{10}, ∞) con el fin de observar la influencia de este valor. En el estudio actual se descartan los resultados obtenidos con la variante `dinámica` debido a que presenta un *trade-off* espacio/tiempo menos competitivo que el obtenido por la variante `estática`.

Texto	E-G$_2$			E-G$_3$			E-G$_4$			E-G$_5$		
	2^6	2^9	2^{10}	2^6	2^9	2^{10}	2^6	2^9	2^{10}	2^6	2^9	2^{10}
WSJ001	**26,22 %**	26,28 %	26,35 %	26,23 %	26,28 %	26,35 %	26,27 %	26,32 %	26,38 %	26,29 %	26,34 %	26,39 %
WSJ010	22,39 %	22,24 %	22,30 %	22,33 %	**22,15 %**	22,20 %	22,34 %	22,16 %	22,21 %	22,35 %	22,16 %	22,21 %
WSJ100	20,37 %	19,85 %	19,86 %	20,24 %	19,64 %	19,63 %	20,18 %	19,57 %	19,55 %	20,17 %	19,55 %	**19,53 %**
FT91	22,57 %	22,38 %	22,43 %	22,48 %	22,25 %	22,28 %	22,42 %	**22,19 %**	22,22 %	22,41 %	**22,19 %**	22,21 %
FT92	20,71 %	20,05 %	20,03 %	20,60 %	19,88 %	19,83 %	20,52 %	19,80 %	19,75 %	20,47 %	19,75 %	**19,70 %**
FT93	19,29 %	18,71 %	18,68 %	19,00 %	18,34 %	18,30 %	18,84 %	18,15 %	18,10 %	18,79 %	18,11 %	**18,06 %**
FT94	19,17 %	18,57 %	18,55 %	18,91 %	18,24 %	18,20 %	18,80 %	18,12 %	18,07 %	18,72 %	18,05 %	**17,99 %**
CR	19,05 %	18,73 %	18,75 %	18,84 %	18,49 %	18,49 %	18,75 %	18,39 %	18,39 %	18,72 %	18,37 %	**18,36 %**

Tabla 5.17: Tasas de compresión obtenidas con E-G$_k$ para la variante `LFU`.

Texto	E-G$_2$			E-G$_3$			E-G$_4$			E-G$_5$		
	2^6	2^9	2^{10}	2^6	2^9	2^{10}	2^6	2^9	2^{10}	2^6	2^9	2^{10}
WSJ001	26,86 %	26,52 %	26,46 %	26,80 %	26,51 %	26,46 %	26,80 %	26,53 %	26,48 %	26,79 %	26,54 %	26,49 %
WSJ010	23,24 %	22,77 %	22,68 %	23,03 %	22,60 %	22,52 %	22,99 %	22,60 %	22,52 %	22,96 %	22,58 %	22,51 %
WSJ100	21,18 %	20,53 %	20,38 %	20,79 %	20,23 %	20,10 %	20,63 %	20,14 %	20,02 %	20,58 %	20,11 %	20,00 %
FT91	23,51 %	22,99 %	22,86 %	23,13 %	22,73 %	22,65 %	23,01 %	22,63 %	22,56 %	22,97 %	22,61 %	22,54 %
FT92	21,35 %	20,72 %	20,57 %	20,99 %	20,47 %	20,33 %	20,83 %	20,37 %	20,25 %	20,75 %	20,30 %	20,19 %
FT93	19,99 %	19,38 %	19,22 %	19,43 %	18,91 %	18,78 %	19,14 %	18,68 %	18,57 %	19,04 %	18,61 %	18,50 %
FT94	19,86 %	19,26 %	19,12 %	19,31 %	18,81 %	18,68 %	19,11 %	18,64 %	18,53 %	18,97 %	18,54 %	18,44 %
CR	19,94 %	19,38 %	19,25 %	19,54 %	19,03 %	18,93 %	19,33 %	18,89 %	18,80 %	19,26 %	18,84 %	18,75 %

Tabla 5.18: Tasas de compresión obtenidas con E-G$_k$ para la variante `dinámica`.

Texto	E-G$_2$	E-G$_3$	E-G$_4$	E-G$_5$
	∞	∞	∞	∞
WSJ001	26,41 %	26,41 %	26,44 %	26,45 %
WSJ010	22,50 %	22,37 %	22,38 %	22,37 %
WSJ100	20,02 %	19,80 %	19,74 %	19,73 %
FT91	22,64 %	22,45 %	22,38 %	22,37 %
FT92	20,16 %	19,99 %	19,93 %	19,88 %
FT93	18,86 %	18,47 %	18,28 %	18,22 %
FT94	18,74 %	18,37 %	18,24 %	18,16 %
CR	18,97 %	18,69 %	18,58 %	18,54 %

Tabla 5.19: Tasas de compresión obtenidas con E-G$_k$ fijando $\alpha = \infty$

La tabla 5.17 presenta las tasas de compresión obtenidas con ordenación/reemplazo `LFU`. Los valores en negrita representan las mejores tasas de compresión obtenidas para cada colección. Como puede observarse, todos estos valores están contenidos en la presente tabla, lo cual demuestra que la estrategia `LFU` es la que mejores resultados obtiene para el presente conjunto de experimentos. Salvo para la colección más pequeña (`WSJ001`), cuyos mejores resultados se obtienen en E-G$_2$ y E-G$_3$, las tasas de compresión mejoran con el número de *rounds* considerados en la identificación de q-gramas. Esto es debido a dos factores complementarios. Por un lado, aumentar el alfabeto de entrada disminuye el número total de símbolos ($|\mathcal{T}|$) necesarios para representar el texto. De la misma forma, al pasar de k a $k+1$ el tamaño del conjunto de aristas ($|\mathcal{A}|$) aumenta de tal forma que las codificaciones NewEdge necesarias para representar las nuevas aristas se amortizan con las codificaciones Follow que las utilizan en las nuevas transiciones identificadas sobre el alfabeto extendido. Sin embargo, esta mejora se reduce paulatinamente con el incremento de k. Se considera $k = 5$ como valor umbral a partir del cual seguir añadiendo q-gramas al alfabeto empeora el *trade-off* espacio/tiempo de la técnica.

Como se anticipaba previamente, aumentar α no mejora notablemente las tasas de compre-

		Compresión				Descompresión				Identif. q-gramas
		2^6	2^9	2^{10}	∞	2^6	2^9	2^{10}	∞	
E-G$_2$	LFU	3,47	3,81	4,10	5,02	0,89	1,11	1,32	2,23	2,06
	Estática	3,48	3,71	3,84		0,80	0,90	1,02		
E-G$_3$	LFU	3,70	4,08	4,25	5,07	0,89	1,11	1,38	1,97	2,25
	Estática	3,71	3,92	4,04		0,77	0,89	0,98		
E-G$_4$	LFU	3,81	4,16	4,35	5,12	0,86	1,12	1,33	1,91	2,38
	Estática	3,81	4,03	4,15		0,88	0,92	1,01		
E-G$_5$	LFU	3,91	4,22	4,43	5,22	0,84	1,12	1,36	1,92	2,46
	Estática	3,91	4,13	4,30		0,75	0,92	1,13		

Tabla 5.20: Tiempos de compresión/descompresión obtenidos con E-G$_k$ para la colección WSJ010.

		Compresión				Descompresión				Identif. q-gramas
		2^6	2^9	2^{10}	∞	2^6	2^9	2^{10}	∞	
E-G$_2$	LFU	113,54	134,72	149,61	390,73	18,15	31,92	43,15	271,12	81,90
	Estática	126,29	133,39	140,63		15,25	21,98	28,66		
E-G$_3$	LFU	129,61	151,52	166,09	372,73	18,71	33,30	46,03	245,39	94,13
	Estática	137,52	146,50	154,88		14,32	23,09	30,29		
E-G$_4$	LFU	137,20	158,39	172,45	369,39	18,71	33,44	44,63	230,79	101,24
	Estática	145,15	153,61	161,96		14,27	22,85	29,77		
E-G$_5$	LFU	141,25	162,06	175,83	367,26	19,81	33,17	45,09	224,33	105,14
	Estática	149,65	157,82	165,60		14,60	23,65	30,49		

Tabla 5.21: Tiempos de compresión/descompresión obtenidos con E-G$_k$ para la colección FT94.

sión obtenidas para los valores considerados. Pasar de $\alpha = 2^9$ a $\alpha = 2^{10}$ no mejora, en ninguno de los casos estudiados, en más de 0,1 puntos porcentuales. Además, para los ficheros pequeños, utilizar $\alpha = 2^9$ puede llegar a mejorar, en algunos casos, los resultados obtenidos con $\alpha = 2^{10}$. Esto nos permite seleccionar $\alpha = 2^9$ como valor de referencia para la presente técnica E-G$_k$ dado que es capaz de mejorar notablemente los tiempos de descompresión obtenidos con $\alpha = 2^{10}$. En términos globales, las tasas de compresión obtenidas alcanzan resultados muy competitivos, bajando del 20 % para colecciones de tamaño medio (como WSJ100) y llegando a valores incluso inferiores al 18 % para colecciones de mayor tamaño (FT93 y FT94).

La tabla 5.18 recoge los resultados obtenidos sobre la variante de ordenación estática de los vocabularios de transiciones de los nodos que representan palabras vacías. Las ratios de compresión obtenidas son entre $0,5$ y 1 puntos porcentuales peores que para el caso anterior. La evolución de los resultados es positiva con el incremento de k y α. Sin embargo, en este último caso, los resultados nunca llegan a los conseguidos para $\alpha = \infty$ (ver tabla 5.19). Estos valores representan el óptimo obtenido por la variante estática aunque son inferiores, en todos los casos, a las mejores tasas conseguidas por LFU sobre un valor finito de α. Aún así, la variante estática es más rápida en descompresión, obteniendo tiempos entre un 30 % y un 50 % menores que para LFU.

La elección de una u otra variante depende, una vez más, del contexto de aplicación en el que vaya a ser utilizada. LFU consigue tasas de compresión mejores que la variante estática a costa de unos tiempos de descompresión menos competitivos como puede verse en las tablas 5.20 y 5.21. Al igual que para E-G$_1$, LFU plantea una menor necesidad de recursos en memoria al trabajar con un modelo limitado a un máximo de α aristas por nodo. Finalmente, la columna Identif. q-gramas, en las tablas 5.20 y 5.21, muestra el tiempo invertido en la etapa de identificación de los q-gramas. Este valor es, en todos los casos (excepto para $\alpha = \infty$) superior al 50 % del tiempo total requerido en el proceso de compresión. Esto supone un importante coste para la técnica E-G$_k$ dado que requiere procesos de compresión intensivos. Sin embargo, en una gran mayoría de los entornos de aplicación, este tiempo es menos importante que el tiempo de descompresión. En el caso actual, podemos observar como la descompresión en E-G$_k$ es hasta 7 veces más rápida que el proceso de compresión. Esto se debe a dos propiedades fundamentales. Por un lado, el compresor requiere un proceso intensivo para la identificación de los q-gramas mientras que el

Texto	E-G$_2$					E-G$_3$				
	LFU		Estática		∞	LFU		Estática		∞
	2^9	2^{10}	2^9	2^{10}		2^9	2^{10}	2^9	2^{10}	
WSJ001	25,76 %	25,81 %	25,94 %	25,92 %	25,87 %	25,74 %	25,78 %	25,92 %	25,89 %	25,84 %
WSJ010	22,62 %	22,65 %	22,99 %	22,88 %	22,77 %	22,57 %	22,59 %	22,90 %	22,82 %	22,69 %
WSJ010	20,62 %	20,56 %	21,08 %	20,95 %	20,61 %	20,50 %	20,45 %	20,94 %	20,83 %	20,49 %
FT91	22,64 %	22,65 %	23,08 %	22,98 %	22,75 %	22,64 %	22,66 %	23,03 %	22,96 %	22,75 %
FT92	20,76 %	20,67 %	21,16 %	21,03 %	20,63 %	20,72 %	20,62 %	21,07 %	20,98 %	20,58 %
FT93	19,16 %	19,06 %	19,49 %	19,38 %	19,02 %	19,09 %	18,99 %	19,39 %	19,30 %	18,94 %
FT94	19,10 %	19,02 %	19,40 %	19,28 %	18,98 %	19,00 %	18,91 %	19,30 %	19,22 %	18,87 %
CR	19,20 %	19,20 %	19,56 %	19,47 %	19,29 %	19,08 %	19,07 %	19,45 %	19,38 %	19,15 %

Tabla 5.22: Tasas de compresión obtenidas con E-G$_k$+ppmdi.

descompresor identifica al alfabeto extendido a partir de la configuración de nodos codificada. Por otro lado, el aumento del valor de k apenas modifica el tiempo de descompresión para un mismo valor de α debido al equilibrio obtenido entre la reducción de $|T|$ y el aumento de $|A|$. Como se ve en la siguiente sección, la eficiencia de E-G$_k$ en descompresión es comparable a otras técnicas estadísticas que obtienen tasas de compresión menos competitivas que las obtenidas por la propuesta actual.

Codificación orientada a byte (E-G$_k$+X). La propuesta actual optimiza el modelo E-G$_k$ para su utilización con una codificación ETDC orientada a byte. Esta implementación se desarrolla sobre el modelo base E-G$_1$ extendido para el manejo dinámico de las relaciones de jerarquía y su codificación mediante el símbolo de escape NewContext identificado como <10000010>.

La descripción conceptual de estas técnicas plantea un proceso en dos etapas: 1) modelado y codificación orientada a byte sobre las propiedades que definen E-G$_k$; 2) codificación del flujo de bytes resultante con una técnica orientada a bit. Esto hace que las técnicas E-G$_k$+X se caractericen por un coste temporal superior a las técnicas E-G$_k$ orientadas a bit con las que comparten la primera etapa. Por lo tanto, el coste añadido para E-G$_k$+X es el asociado a los procesos de compresión/descompresión del flujo de bytes sobre la técnica de orden superior X utilizada en la segunda etapa. De esta manera, las técnicas E-G$_k$+X están obligadas a obtener una mejor efectividad que la conseguida por las diferentes variantes de E-G$_k$ (sobre codificación de Huffman) para plantearse como alternativas a éstas. Para el caso base, E-G$_1$+ppmdi obtenía mejores tasas de compresión. Sin embargo, no sucede lo mismo para el caso actual.

La tabla 5.22 muestra las tasas de compresión obtenidas por E-G$_2$+ppmdi y E-G$_3$+ppmdi sobre las variantes LFU y estática. Comparando estos resultados con los obtenidos por E-G$_k$ (ver tablas 5.17, 5.18 y 5.19) podemos observar como los actuales son hasta 1 punto porcentual peores que los obtenidos en el caso anterior (excepto para WSJ001). Además, la técnica actual apenas aprovecha las propiedades obtenidas al aumentar el valor de k como puede observarse en la pequeña mejora obtenida por E-G$_3$+ppmdi respecto a E-G$_2$+ppmdi. Sin embargo, E-G$_2$+ppmdi si muestra una notable mejoría respecto al caso base motivada por la mayor significatividad del los bigramas obtenidos en el primer *round*. Los resultados obtenidos por E-G$_k$+bzip2 no superan, en ningún caso, los obtenidos por E-G$_k$, por lo que se descartan al no ser competitivos con los previamente presentados.

5.5.4. Comparativa

Finalmente, en esta sección se comparan los resultados conseguidos por las técnicas basadas en el modelo E-G respecto a los obtenidos con otras propuestas comparables en el entorno actual. La decisión de que técnicas considerar para esta comparación ha sido complicada debido a la dificultad encontrada para la localización de implementaciones reales de compresores de orden superior y orientados a palabras.

La tabla 5.23 contiene una selección de resultados obtenidos sobre las técnicas E-G$_1$ y E-G$_k$ (se descartan los obtenidos sobre codificación orientada a byte al ser menos competitivos en el contexto actual). Para cada texto considerado se seleccionan dos resultados para E-G$_1$ y otros dos para E-G$_k$. En cada caso se selecciona la configuración que permite obtener la mejor tasa de compresión (variante LFU) y la que se caracteriza por una mejor eficiencia temporal en los procesos de compresión y descompresión (variante estática). El valor de k se selecciona en el intervalo $[2^8, 2^{10}]$ considerando que éste contiene los valores de α para los que las respectivas técnicas muestran un comportamiento más competitivo.

Cada una de estas variantes se refiere como E-G$_{k;\alpha}$, donde k y α se interpretan de la manera referida al principio del capítulo actual. Para E-G$_1$ la mejor compresión se obtienen siempre con la variante LFU y $\alpha = 2^{10}$, mientras que los mejores tiempos se consiguen con la variante estática y $\alpha = 2^9$. Por su parte, las configuraciones más eficiente para E-G$_k$ se obtienen con E-G$_2$ y $\alpha = 2^9$, mientras que las más efectivas varían para cada colección. Los datos mostrados en esta tabla se complementan con la figura 5.13 que representa los tiempos de compresión y descompresión obtenidos por las configuraciones de E-G$_1$ y E-G$_k$ obtenidas para obtener los resultados que se presentan en la tabla actual.

	AP001	ZIFF010	FT91	AP020	WSJ040	CR	ZIFF060	AP100	FT92	FT93	FT94
	1,13	10,65	14,07	20,24	40,69	48,72	60,05	100,15	167,32	188,43	194,34
Comp.	27,61 %	22,75 %	23,42 %	22,96 %	22,19 %	20,30 %	21,75 %	21,99 %	21,54 %	20,73 %	20,62 %
	E-G$_{1;2^{10}}$										
Tiempo	27,91 %	23,35 %	23,98 %	23,52 %	22,75 %	20,84 %	22,43 %	22,65 %	22,22 %	21,37 %	21,27 %
	E-G$_{1;2^9}$										

	AP001	ZIFF010	FT91	AP020	WSJ040	CR	ZIFF060	AP100	FT92	FT93	FT94
	1,13	10,65	14,07	20,24	40,69	48,72	60,05	100,15	167,32	188,43	194,34
Comp.	26,78 %	21,24 %	22,19 %	21,33 %	20,48 %	18,36 %	19,77 %	19,92 %	19,70 %	18,06 %	17,99 %
	E-G$_{4;2^9}$	E-G$_{5;2^9}$	E-G$_{4;2^9}$	E-G$_{4;2^{10}}$		E-G$_{5;2^{10}}$	E-G$_{5;2^9}$	E-G$_{5;2^{10}}$			
Tiempo	27,14 %	22,07 %	22,99 %	22,23 %	21,38 %	19,38 %	20,81 %	20,96 %	20,72 %	19,38 %	19,26 %
	E-G$_{2;2^9}$										

Tabla 5.23: Selección de tasas de compresión obtenidas con E-G$_1$ y E-G$_k$.

La configuración de la tabla 5.23 muestra como las ratios de compresión de las variantes más efectiva y más eficiente se alejan con el incremento del tamaño del texto. Esta diferencia es menor para E-G$_1$ que para E-G$_k$ donde llega a alcanzar los 2,27 puntos porcentuales en la colección FT94. Sin embargo, como se verá más adelante, en estos casos los tiempos de cómputo presenta un mayor desequilibrio en favor de la variante estática. Los resultados obtenidos para las colecciones de mayor tamaño presentan un intervalo de efectividad entre el 19, 70 % y 22, 22 % para FT92 y entre el 17, 99 % y 21, 27 % para FT94. Esto muestra la flexibilidad de las presentes técnicas E-G para adaptar su *trade-off* espacio/tiempo a las propiedades específicas de su contexto de aplicación. Esta afirmación se respalda, a continuación, sobre el análisis de las propiedades temporales de estas técnicas.

Las tablas 5.24, 5.25 y 5.26 resumen las tasas de compresión obtenidas por diferentes tipos de compresores. Nótese la dificultad de establecer un conjunto de comparación más específico para compresores de orden superior orientados a palabras ya que no ha sido posible acceder a ninguna implementación funcional de las técnicas planteadas en la §5.1. A excepción de la técnica presentada en [ÁCPFL09], que obtiene unas ratios de compresión entre el 28 % y el 31 %, el resto de las técnicas referidas concluyen indicando su ligera mejoría respecto a la compresión PPM. En todos los casos, los resultados son peores que los obtenidos por nuestras técnicas, dado que éstas mejoran notablemente los resultados obtenidos por ppmdi.

La tabla 5.24 revisa la efectividad de varios compresores universales. Todos ellos comparten la utilización de un alfabeto de entrada orientado a caracteres. Por un lado se consideran técnicas

	AP001 1,13	ZIFF010 10,65	FT91 14,07	AP020 20,24	WSJ040 40,69	CR 48,72	ZIFF060 60,05	AP100 100,15	FT92 167,32	FT93 188,43	FT94 194,34
bzip2	28,31 %	25,00 %	27,06 %	27,11 %	26,98 %	24,14 %	25,13 %	27,23 %	27,10 %	25,32 %	25,27 %
gzip	37,63 %	33,01 %	36,42 %	37,14 %	37,22 %	33,29 %	33,11 %	37,34 %	36,48 %	34,21 %	34,21 %
p7zip	28,94 %	24,60 %	26,11 %	**24,45 %**	25,61 %	22,93 %	24,39 %	**24,21 %**	25,55 %	23,82 %	23,75 %
ppmdi	**26,43 %**	**22,97 %**	**25,30** %	25,52 %	**25,51 %**	**22,42 %**	**23,12 %**	25,64 %	**25,34 %**	**23,55 %**	**23,55 %**

Tabla 5.24: Tasas de compresión obtenidas por diferentes compresores universales.

	AP001 1,13	ZIFF010 10,65	FT91 14,07	AP020 20,24	WSJ040 40,69	CR 48,72	ZIFF060 60,05	AP100 100,15	FT92 167,32	FT93 188,43	FT94 194,34
mPPM	**24,44 %**	21,35 %	22,75 %	22,53 %	22,68 %	19,97 %	21,28 %	22,47 %	22,56 %	20,86 %	20,91 %
WCIM	24,88 %	**21,07 %**	**22,28 %**	**21,89 %**	**21,97 %**	**19,48 %**	**20,76 %**	**21,57 %**	**21,70 %**	**20,07 %**	**20,11 %**
ETDC	26,30 %	21,44 %	22,86 %	22,15 %	22,17 %	19,71 %	20,92 %	21,66 %	21,88 %	20,26 %	20,29 %

Tabla 5.25: Tasas de compresión obtenidas por técnicas de mapeo palabra-código sobre ppmdi.

	AP001 1,13	ZIFF010 10,65	FT91 14,07	AP020 20,24	WSJ040 40,69	CR 48,72	ZIFF060 60,05	AP100 100,15	FT92 167,32	FT93 188,43	FT94 194,34
v2vdc	32,16 %	26,30 %	26,99 %	26,98 %	25,69 %	23,47 %	24,96 %	25,34 %	24,49 %	22,94 %	22,84 %
v2vdc$_H$	32,07 %	26,00 %	26,65 %	26,65 %	25,36 %	23,13 %	24,58 %	24,96 %	24,07 %	**22,56 %**	**22,47 %**
Re-Pair	**29,04 %**	**23,02 %**	**24,00 %**	**22,87 %**	**22,44 %**	**20,16 %**	**21,26 %**	**21,14 %**	**21,34 %**	-	-

Tabla 5.26: Tasas de compresión obtenidas por diferentes compresores orientados a frases.

basadas en los algoritmos LZ: gzip es un compresor clásico mientras que p7zip se basa en una revisión del algoritmo LZ77 denominada lzma. Por otro lado, bzip2 y ppmdi son compresores de orden superior desarrollados, respectivamente, sobre la BWT y el modelo de contextos PPM. Todos los compresores se ejecutan en su configuración por defecto (ver Apéndice A). ppmdi obtiene los mejores resultados para todas las colecciones excepto para ap020 y ap100 en las que p7zip es la mejor opción. Estas dos técnicas son las más efectivas dado que ni gzip ni bzip2 son capaces de mejorar sus resultados para ninguna de las colecciones. Sin embargo, estas dos son las que mantienen un mejor comportamiento temporal como se observa más adelante. ppmdi sólo mejora la efectividad de las técnicas E-G para el texto de menor tamaño (ap001). Para el resto de los casos tanto E-G$_1$ como E-G$_k$ superan notablemente a ppmdi hasta una diferencia máxima superior a los $5,5$ puntos porcentuales para FT94.

Por su parte, la tabla 5.25 ofrece un resumen de las tasas de compresión obtenidas con técnicas de mapeo palabra-código sobre ppmdi. Este conjunto de compresores ha sido profundamente analizado en el capítulo anterior y se caracteriza por obtener una compresión de orden superior del texto basada en la recompresión de su flujo de salida orientado a byte con un codificador orientado a bit. Estas técnicas mejoran notablemente la efectividad de los compresores previamente analizados. La técnica WCIM (presentada en el capítulo anterior) se muestra como la más efectiva, mejorando ligeramente las ratios de compresión obtenidas por la combinación de ETDC+ppmdi que, a su vez, presenta una mejor eficiencia temporal. Finalmente, la técnica adaptativa mPPM supone el umbral de comparación entre estás técnicas y nuestra propuesta E-G$_1$ que muestra una efectividad muy cercana a la obtenida por mPPM pero inferior a las otras dos técnicas. Por su parte, la efectividad de E-G$_k$ supera notablemente la obtenida por estas técnicas excepto, una vez más, para textos de pequeño tamaño. La diferencia crece con el tamaño del texto superando los dos puntos porcentuales para las colecciones de mayor tamaño.

Finalmente, la tabla 5.26 muestra los resultados obtenidos por sendas técnicas de compresión orientadas a frases y que, por tanto, manejan un alfabeto de palabras extendido con secuencias al igual que lo hace E-G$_k$. Las dos primeras filas presentan los resultados de la técnica v2vdc en su variante general y en la construida sobre la heurística basada en *ganancia* para la obtención de las frases (v2vdc$_H$). En ambos casos, al igual que en el trabajo original [BFL$^+$10], se utiliza $minFreq = 10$ lo que exige que la frase ocurra, al menos, 10 veces en el texto. Por su parte,

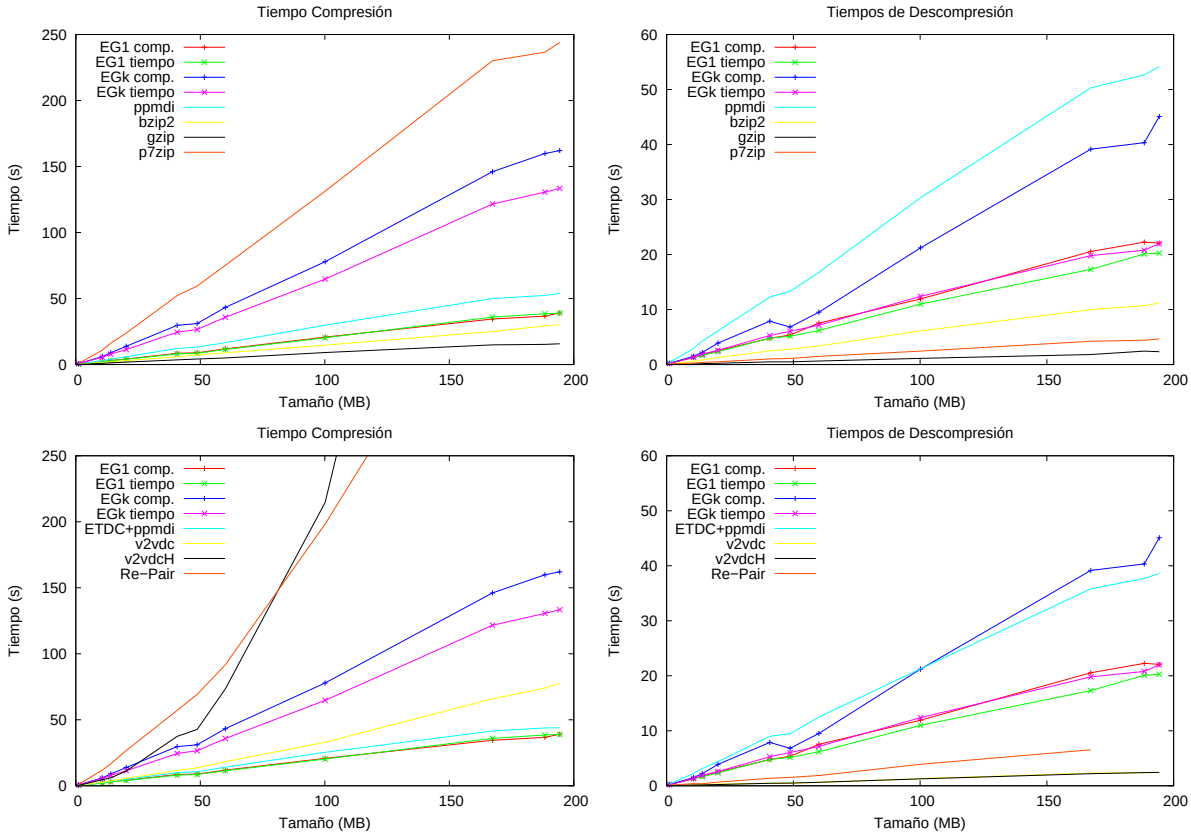


Figura 5.13: Evolución de los tiempos de compresión/descompresión dentro del presente entorno de experimentación.

la fila `Re-Pair` contiene los resultados obtenidos al aplicar la implementación de Raymond Wan[5] a los textos seleccionado y comprimir la secuencia $\mathcal{S}$ con un código de `Huffman`. Para las colecciones `FT93` y `FT94` no se muestra resultado ya que los experimentos no han finalizado dados los requisitos de memoria que plantea la técnica para estos tamaños de texto. Como puede observarse, `Re-Pair` es siempre la mejor opción, aunque debe considerarse que `v2vdc` utiliza una codificación `ETDC` orientada a byte. Considerando que `E-G`$_k$ adapta el algoritmo `Re-Pair` para la identificación de q-gramas, es interesante reseñar que nuestra propuesta mejora siempre el algoritmo original en más de un punto porcentual.

Este estudio deja patente que la efectividad de la técnica `E-G`$_k$ es la más competitiva en el presente entorno de experimentación excepto para los ficheros de pequeño tamaño. La figura 5.13 complementa el estudio anterior con un análisis temporal de las técnicas consideradas. Este estudio contempla los tiempos de compresión/descompresión obtenidos por las configuraciones de `E-G`$_1$ y `E-G`$_k$ analizadas en la tabla 5.23. Por tanto, los tiempos representados corresponden a las configuraciones más efectivas (referidas como `E-G`$_1$ `comp.` y `E-G`$_k$ `comp.`) y más eficientes (referidas como `E-G`$_1$ `tiempo` y `E-G`$_k$ `tiempo`). Las figuras superiores comparan los tiempos de compresión y descompresión de nuestras técnicas respecto a los compresores universales referidos, mientras que las figuras inferiores comparan `E-G`$_1$ y `E-G`$_k$ respecto a la combinación `ETDC+ppmdi` (cuyos tiempos de compresión y descompresión mejoran los obtenidos por las técnicas equivalentes) y las técnicas orientadas a frases. El eje `X`, en cada figura, representa el tamaño (en MB.) de los textos considerados en la experimentación.

[5] `http://www.cbrc.jp/~rwan/en/restore.html`

La comparativa respecto a los compresores universales muestra como `gzip` y `bzip2` plantean las opciones más eficientes en tiempos de compresión, mientras que `p7zip` se sitúa entre las dos propuestas anteriores en lo que respecta a tiempos de descompresión. Por su parte, las dos configuraciones de `E-G`$_1$ necesitan unos tiempos de compresión menores que los de `ppmdi`, mientras que los obtenidos por `E-G`$_k$ sólo son inferiores a los requeridos por `p7zip`. En este caso, la eficiencia temporal de `E-G`$_k$ se ve claramente degradada por la etapa de identificación de q-gramas previa al propio proceso de compresión. Sin embargo, en descompresión, `E-G`$_k$ se postula como una alternativa competitiva capaz de mejorar la eficiencia de `ppmdi` y acercándose notablemente a los tiempos de compresión obtenidos por `E-G`$_1$. Puede observarse como la configuración más eficiente de `E-G`$_k$ es capaz de mejorar los tiempos de descompresión de la configuración más efectiva de `E-G`$_1$, obteniendo, a su vez, unas mejores ratios de compresión que esta última como puede comprobarse en la tabla 5.23.

Las dos figuras inferiores muestran la comparativa obtenida respecto a `ETDC+ppmdi` y los compresores orientados a frases. Puede observarse como las dos configuraciones consideradas de `E-G`$_1$ son las opciones más eficientes, en tiempo de compresión, junto con la técnica `ETDC+ppmdi`. En lo que respecta a los compresores orientados a frases, `v2vdc` requiere un menor tiempo de compresión que `E-G`$_k$ mientras que `v2vdc`$_H$ y `Re-Pair` se disparan para textos de mayor tamaño. Los tiempos de descompresión de `E-G`$_1$ y `E-G`$_k$ se ven superados por los obtenidos por `v2vdc` y `Re-Pair` cuyo modelado de orden 0 posibilita la alta eficiencia obtenida en sus respectivos procesos de descompresión. Finalmente, puede observarse como nuestras propuestas (excepto `E-G`$_k$ `espacio`) son capaces de mejorar los tiempos de descompresión obtenidos por `ETDC+ppmdi` cuya propiedad distintitiva es su capacidad de "acelerar" al propio `ppmdi`.

5.6 Conclusiones y trabajo futuro

La familia de compresores `Edge-Guided (E-G)` se construye sobre un modelo del texto orientado a la representación de las relaciones de adyacencia existente entre sus símbolos constituyentes. La propuesta base `E-G`$_1$ considera un alfabeto de entrada orientado a palabras mientras que su generalización `E-G`$_k$ extiende el alfabeto original con un conjunto de q-gramas significativos obtenidos sobre una variante del algoritmo `Re-Pair`. Esto supone la construcción de sendos modelos de orden 1, orientado a palabras para `E-G`$_1$ y orientado a secuencias de palabras para `E-G`$_k$. Este último resultado puede plantearse también como un modelo basado en estadísticas de orden superior, al considerar el concepto original de palabra como unidad mínima constituyente de las secuencias finalmente modeladas como símbolos del modelo.

El problema principal al que se enfrentan este tipo de técnicas radica en la codificación del modelo utilizado para la propia codificación del texto. Las propuestas `E-G` combinan un modelado estadístico (basado en las propiedades de las relaciones de adyacencia) con una codificación de diccionario planteada de forma distribuida en cada uno de los nodos que componen el grafo utilizado como modelo. La gestión de estos diccionarios supone un reto a la hora de establecer políticas de organización capaces de obtener una *trade-off* espacio/tiempo competitivo. Las variantes propuestas consideran, por un lado, operar con una versión reducida del modelo, limitando el grado de salida máximo de cada nodo. De esta manera se evita la utilización de palabras de código de gran longitud para la codificación de los vocabularios de transiciones asociados a las palabras vacías. Por otro lado, se plantea la posibilidad de utilizar una representación completa del texto en la que las palabras vacías se "penalizan" de tal forma que su gestión no requiera un tiempo de cómputo inabordable. Los resultados obtenidos demuestran que ambas variantes son competitivas para diferentes contextos de aplicación.

El esquema diseñado para `E-G`$_1$, sobre codificación de `Huffman`, muestra una alta efectividad en el contexto estudiado. Además, se adapta fácilmente para la codificación del resultado obte-

nido con E-G$_k$ aprovechando mejor la nueva representación del texto obtenida sobre el alfabeto extendido con los q-gramas. En términos globales, las técnicas E-G$_k$ son poco competitivas para la compresión de textos de pequeño tamaño (debido a la sobrecarga inherente a la codificación del modelo). Para textos de mediano y gran tamaño, los resultados obtenidos se sitúan en el rango [18 %, 22 %] mejorando todas aquellas soluciones comparables analizadas en la sección anterior. En lo que respecta a la eficiencia, E-G$_1$ posee un comportamiento interesante para compresión y descompresión, considerando que se desarrolla sobre una modelado adaptativo del texto. Por su parte, E-G$_k$ se caracteriza por una descompresión rápida supeditada a procesos de compresión intensivos debido a la necesidad de identificar los q-gramas significativos con los que extiende su alfabeto de entrada. Aún así, la descompresión E-G$_k$ no se ve perjudicada por valores de k crecientes y mantiene unos tiempos medios capaces de mejorar los obtenidos por técnicas de orden superior orientadas a caracteres como `ppmdi` o combinaciones de técnicas como `ETDC+ppmdi`.

Lo resultados obtenidos por las técnicas E-G$_k$ suponen, a su vez, una demostración de la bondad de la propuesta jerárquica utilizada para la representación de los q-gramas utilizados en el alfabeto de entrada. Esta variante permite obtener gramáticas libres de contexto de menor tamaño que las generadas con el algoritmo `Re-Pair` original. Esto supone disponer de una representación más significativa del texto de acuerdo tanto a la gramática como a la propia secuencia comprimida. El modelado construido en E-G$_k$ mejora notablemente la representación obtenida con E-G$_1$ gracias a la jerarquía de derivación de contextos diseñada sobre la descomposión de la gramática en "heavy paths". Estos resultados suponen, en términos generales, que el uso de una gramática libre de contexto hace viable el diseño de un modelo del lenguaje de orden superior y orientado a palabras sobre una configuración capaz de captar grandes cantidades de información con un número limitado de reglas.

Finalmente, el grafo E-G propuesto como base de las técnicas actuales es capaz de obtener una representación significativa de la información contenida en el texto como ha quedado demostrado en las tasas de compresión obtenidas por las diferentes técnicas propuestas. Esto se puede aprovechar en otro tipo de aplicaciones relacionadas, genéricamente, con la extracción de información de un texto. En [MPAdlF08] se plantea una idea inicial para la extracción de términos descriptivos de un texto utilizando una versión primitiva del actual modelo E-G$_k$. Los primeros resultados obtenidos sobre la identificación de q-gramas significativos sugieren que el modelo representa una base de conocimiento válida para extracción de este tipo de términos. Sin embargo, la variante actual resulta excesivamente genérica para este propósito y el análisis de resultados requiere más intervención humana de la que sería deseable para un sistema semi-automático. El algoritmo `TextRank` [MS05] propone una política de ranking construida de acuerdo a la importancia de los nodos que forman el modelo del texto obtenido. La "votación" de un nodo se lleva a cabo sobre la estructura de aristas que lo caracteriza. Esto permite obtener una valoración de la importancia que éste posee por si mismo junto con la derivada de su estructura de relaciones en el grafo. Cómo incorporar esta experiencia en la definición de pares activos para nuestro proceso de identificación de q-gramas significativos supone un interesante reto futuro enfocado en la utilización del modelo E-G en procesos de extracción de términos clave en un texto, facilitando los procesos de catalogación de sus contenidos.

No importa el problema
(no) importa la solución.
Andrés Calamaro

6

Compresión de Corpus Paralelos Bilingües

La gran riqueza cultural de la sociedad humana conlleva la publicación de una amplia variedad de contenidos en diferentes idiomas. El volumen de estas publicaciones se ha disparado en los últimos años gracias a la generalización en el uso de los nuevos recursos electrónicos de publicación existentes en la WWW.

Un ejemplo claro de este hecho lo representa España donde existen cuatro idiomas oficiales: castellano, catalán, gallego y vasco. Esto supone la convivencia diaria de personas cuya comunicación se lleva a cabo en diferentes idiomas dentro de un mismo ámbito territorial en el que se demanda la publicación regular de diversos contenidos en cada uno de estos idiomas. Otros países (como Canadá, Bélgica o Irlanda) o entidades supranacionales (como la Unión Europea) comparten la situación anterior. Además, el desarrollo y asentamiento de redes de comunicación globales, como la WWW, ha favorecido la creación de grandes proyectos de índole internacional y, por tanto, tendentes a la formación de un entorno multicultural que alberga la demanda de información publicada en diferentes idiomas. Esto trae consigo la creación y desarrollo de grandes bases de datos de texto, de carácter multilingüe, que necesitan un conjunto específico de recursos con el que prestar un servicio eficaz a sus usuarios.

El trabajo presentado en este capítulo se desarrolla, sobre el contexto anterior, a través del concepto de *corpus paralelo bilingüe* (**bitexto**). Un *bitexto* es una entidad formada por dos textos (*origen* y *meta*) que son traducción mutua. En otras palabras, los dos textos que forman un bitexto contienen la misma información en dos idiomas diferentes. Sin embargo, esta afirmación es rebatible en términos prácticos dado que la información contenida en el bitexto no depende exclusivamente de su *significado* sino que también existe una información asociada al *estilo* del traductor que, a veces, no es tan marginal como se esperaría desde un punto de vista teórico [NMB92].

Melamed [Mel01] considera los bitextos como "*una de las fuentes de conocimiento lingüístico más importantes dado que la traducción de un texto en otro idioma puede ser observada como una anotación detallada del significado de dicho texto*". La creciente disponibilidad de grandes colecciones de estos bitextos ha favorecido, el desarrollo y asentamiento de diferentes campos de investigación y aplicación dentro del área de procesamiento de lenguaje natural. Algunas de estas aplicaciones utilizan más de un idioma para su funcionamiento – por ejemplo aplicaciones de traducción automática [CW03, Lop08], o recuperación de información translingüe [GF04] – o, por el contrario, son aplicaciones monolingües – análisis sintáctico [Car03] o desambiguación del significado de las palabras [IV98] – que utilizan los bitextos para proyectar, sobre el idioma objetivo, el conocimiento lingüístico contenido en el idioma complemantario [MS05].

La caracterización dada de bitexto invita a pensar en obtener una representación "comprimida" del mismo, considerando que está compuesto por sendas versiones del mismo contenido expresadas, cada una de ellas, en un determinado idioma. Esto implica la existencia de una alta redundancia semántica con las consiguientes desventajas que esto supone, tanto en lo que respecta al espacio ocupado como a la infrautilización de la relación semántica existente entre cada uno de los textos.

El presente capítulo plantea diferentes técnicas para afrontar este problema. A pesar del interés suscitado ante el uso de bitextos, en aplicaciones como las planteadas anteriormente, existe muy poco trabajo de investigación centrado en la representación compacta de este tipo de textos. Cómo se muestra en la Sección §6.1, apenas se han localizado un par de trabajos centrados en la compresión de bitextos y ninguno de ellos trata la posibilidad de operar directamente con las representaciones comprimidas.

Nuestro objetivo es estudiar la naturaleza estadística de los bitextos y plantear diferentes modelos de representación orientados a su uso posterior en el campo de la compresión. Para ello, nos centraremos en las propiedades semánticas específicas de este tipo de textos sin obviar que su contenido responde a las propiedades de los textos en lenguaje natural consideradas en los capítulos previos. Esto supone, en primer lugar, que el modelado natural de bitextos comparte las propiedades consideradas para textos genéricos en lenguaje natural y, por tanto, se puede afrontar desde una perspectiva orientada a palabras. Sin embargo, al acometer el problema desde un punto de vista genérico no se considera la relación de traducción existente entre diferentes fragmentos de cada texto del bitexto. Consideraremos un *baseline* general que no maneja directamente la relación de traducción existente en el bitexto. Esta decisión permite establecer un límite superior para la efectividad esperada de nuestras propuestas.

A partir de esta experiencia inicial, consideraremos la información recogida en los alineamientos textuales (ver sección siguiente) para relacionar aquellos fragmentos de cada texto que son traducción mutua en el bitexto. Esta transformación mantiene la información original del bitexto y sobre ella añade el conocimiento necesario para utilizar la relación de traducción contenida en él. Sobre este escenario se considera la posibilidad de evolucionar el *baseline* anterior con el fin de obtener un nuevo modelo orientado a palabras capaz de representar la relación de traducción existente entre los textos. Esta premisa se utiliza como base para el desarrollo de una propuesta de orden 1 en la que las palabras identificadas en el lenguaje origen se usan como contexto para la codificación de sus "traducciones" en el lenguaje meta. Finalmente, se considera la definición de un nueva entidad simbólica, denominada **bipalabra** [MPASM$^+$09], capaz de conjugar en una entidad única las propiedades derivadas del lenguaje natural con las relaciones de traducción obtenidas gracias a la transformación basada en alineamiento textual. La técnica `2LCAB` (*2-Level Compressor for Aligned Bitexts*) [ABMPSM09], orientada a bipalabras, supone un importante paso adelante ya que permite el acceso y la búsqueda directa en las representaciones comprimidas obtenidas como resultado. Estas propiedades se utilizan para la implementación eficiente de operaciones monolingües y translingües (entendiendo como tal aquellas operaciones cuyo resultado se obtiene en el idioma complementario al utilizado en la consulta) interesantes en el contexto de aplicación anteriormente expuesto.

La organización del capítulo actual presenta, en primer lugar (§6.1), una revisión del trabajo relacionado con la propuesta actual, incidiendo en las propuestas desarrolladas para la compresión de bitextos. La Sección §6.2 introduce los conceptos básicos asociados con las técnicas de alineamiento textual utilizadas en la transformación del bitexto. Las Secciones §6.3 y §6.4 detallan, respectivamente, las propuestas planteadas para el modelado de bitextos y su utilización en diferentes técnicas de compresión capaces de aprovechar la información contenida en el modelo de acuerdo a unas objetivos particulares. Finalmente, la Sección §6.5 profundiza en las propiedades estadísticas de los bitextos y describe los resultados obtenidos por las propuestas anteriores dentro de un marco experimental constituido a partir de un conjunto heterogéneo de bitextos. La sección de conclusiones y trabajo futuro (§6.6) plantea una evaluación crítica del capítulo, centrándose en cómo los resultados obtenidos se pueden incorporar directamente en aplicaciones de búsqueda de concordancias bilingües y, a su vez, como el conocimiento derivado del estudio actual puede ser aplicado en otras aplicaciones relacionadas con el uso de bitextos y su representación compacta.

6.1 Trabajo relacionado

Comprimir independientemente un texto en un idioma y su traducción en otro diferente plantea una operación inefectiva dado que una parte considerable de la información contenida en el texto meta es altamente redundante al representar la misma semántica que el texto origen. Esto incrementa innecesariamente los costes de almacenamiento y transmisión del bitexto al mantener diferentes versiones del mismo contenido. Idealmente esto se podría evitar si la segunda (y sucesivas) versiones se pudiesen obtener a partir de su traducción en el texto origen.

La caracterización sobre la que desarrolla el problema de la compresión de bitextos fija su objetivo en obtener una representación que mejore tanto su almacenamiento como su transmisión [NMB92] aprovechando la relación semántica existente entre sus textos componentes. Además, esta propiedad permite diferir la decisión de qué idioma es utilizado para mostrar una determinada información considerando, incluso, la posibilidad de mantener una presentación simultánea en los diferentes idiomas.

Sobre esta base, Nevill-Manning y Bell [NMB92] presentan, en 1992, el primer trabajo centrado en la compresión de bitextos. Los autores consideran que, en general, los bitextos son textos en lenguaje natural que comparten un contenido semántico expresado en diferentes idiomas. Desde una perspectiva teórica, las versiones traducidas de un texto original no contienen mucha información extra respecto a dicho original, lo que debería facilitar la obtención de una representación equivalente del bitexto con un pequeño coste. Este primer trabajo investiga como los métodos clásicos de compresión pueden ser extendidos para afrontar el problema actual.

Una primera etapa trata de evaluar la cantidad de información extra que aporta el texto meta respecto al original. Para ello se plantea un estudio, con usuarios humanos, consistente en predecir los siguientes símbolos de un texto con y sin la ayuda de un texto que contenga la misma información en un idioma diferente (referido como *texto paralelo*). El fundamento del presente experimento es que un humano, conocedor de los idiomas envueltos en dicho experimento, predice con mayor facilidad el contenido del texto original utilizando la información contextual aportada por el texto paralelo. Los resultados obtenidos, sobre tres textos diferentes, confirman las expectativas iniciales obteniendo una entropía menor para las predicciones realizadas con la aportación del texto paralelo. Sin embargo, las conclusiones obtenidas del estudio plantean que la utilidad del texto paralelo es variable y que la cantidad de información extra contenida en la traducción del original es sorprendentemente grande en algunos casos. Esta propiedad es muy interesante, para el caso actual, dado que supone una indicación de cuánto de *significado* y de *estilo*, asociado al traductor, hay en el bitexto.

El desarrollo del compresor, en la segunda etapa, se lleva a cabo sobre la experiencia anterior. Esto supone disponer de un conocimiento de los dos idiomas contenidos en el bitexto además de un conjunto de "pistas" derivadas a partir del texto paralelo. Un humano utiliza el texto paralelo cuando éste plantea una buena sugerencia para la predicción del original. En otro caso, utiliza el estilo y el significado ya extraído para la predicción del siguiente símbolo. El compresor propuesto adopta estas mismas propiedades para la implementación de su modelo. Esto supone combinar el uso de un modelo general y otro específico sobre la información derivada del texto paralelo. El compresor contempla una operación de mezclado (*blending*) entre ambos modelos imitando el comportamiento humano anteriormente descrito. Para el modelo general se desarrolla una propuesta basada en `PPM` [CW84b] mientras que para las predicciones basadas en la semántica del contexto proponen un modelo desarrollado sobre dos relaciones específicas de los bitextos: palabras *coincidentes* en ambos textos y *sinónimos* obtenidos mediante un tesauro. El funcionamiento global del algoritmo se basa en codificar el texto origen de acuerdo a sus propiedades particulares y el texto meta mediante la combinación de las predicciones obtenidas a partir del texto paralelo y las obtenidas sobre el modelo `PPM`. Las predicciones se ponderan de

acuerdo a sendos valores de pesado. La elección de estos valores determina la efectividad de la técnica. Los resultados presentados muestran cómo esta nueva técnica mejora notablemente la entropía de la fuente obtenida sobre una modelo predictivo clásico.

El siguiente trabajo relevante, en este área de investigación, se produce tras un notable vacío de resultados desde la publicación del trabajo anterior. Conley y Klein [CK08] plantean una nueva propuesta con una perspectiva basada en los avances obtenidos en procesamiento de lenguaje natural en los dieciséis años transcurridos desde la propuesta anterior. Nevill-Manning y Bell citan, varias veces, las posibles mejoras que podrían obtenerse en la compresión de textos paralelos con la utilización de técnicas de traducción automática. Para esta nueva propuesta, los autores consideran los avances desarrollados en lo que respecta a técnicas de alineamiento textual. Éstas se utilizan con el propósito de obtener una transformación del bitexto en la que se pueden identificar las relaciones semánticas existentes entre los dos textos que lo forman.

Sobre esta transformación del texto se propone una técnica basada en diccionario en la que el texto origen se codifica de acuerdo a sus propiedades y además se usa como diccionario para la representación del texto meta. Los símbolos originales del texto meta se pueden recuperar utilizando un glosario bilingüe y otros elementos lingüísticos que se comentan más adelante. La traducción de una unidad del texto meta se localiza en una ventana de pequeño tamaño (en el texto origen) que abarca una frase o, a lo sumo, un párrafo, excepto en aquellos casos en los que alguna de los textos tenga información omitida respecto al otro.

El presente algoritmo de compresión, denominado Trans, se ejecuta sobre un conjunto de recursos lingüísticos formado por los textos origen y meta, los alineamientos (a nivel de palabra y frase) considerados entre los dos textos, sendas formas lematizadas de los dos textos, un diccionario de lemas sobre el texto origen y otro de variantes sobre el meta y, finalmente, un glosario bilingüe para el par de textos utilizado. A partir de toda esta infraestructura lingüística, Trans considera tres árboles de `Huffman` (H_1, H_2 y H_3) para codificar las distintas partes que conforman el resultado del algoritmo: H_1 se utiliza para la codificación de todas las palabras no alineadas en el bitexto; H_2 se responsabiliza de codificar el desplazamiento de cada palabra en el texto meta utilizando como referencia la posición esperada en el origen; finalmente, el tercer árbol de `Huffman`, H_3, codifica el resto de valores que se distribuye en un rango de enteros menor que el anterior dado que, a priori, el número de variantes para una palabra determinada es pequeño.

Una vez caracterizada la técnica, los autores muestran unos resultados experimentales cuya interpretación es bastante difusa dado que en las ratios de compresión obtenidas no consideran el tamaño de los ficheros auxiliares que Trans necesita en la etapa de descompresión. Aún así, la efectividad obtenida en la compresión del texto meta es sólo ligeramente mejor a la conseguida por `bzip2` (0,2 puntos porcentuales para el corpus inglés-francés utilizado). Las conclusiones finales del trabajo plantean un par de ideas interesantes centradas, por un lado, en la extensión del modelo Trans de tal forma que permita la compresión de $k-1$ textos meta respecto a un texto original y, por otro lado, en el análisis de oportunidades, de la actual propuesta Trans, de cara a soportar operaciones de búsqueda sobre el resultado comprimido. Esta última idea se afronta directamente en el capítulo actual (ver propuesta `2LCAB` en la Sección §6.4.4) obteniendo unos resultados interesantes fácilmente explotables en contextos de aplicación prácticos relacionados con la búsqueda de concordancias bilingües.

6.2 Conceptos básicos

La propiedad principal de un bitexto radica en la relación de traducción mutua existente entre sus textos componentes. Esto supone que las aplicaciones que los utilizan necesitan una determinada representación que facilite identificar y aprovechar dicha relación de traducción de acuerdo

a sus objetivos específicos. Para este propósito se ejecutan procesos de *alineamiento textual*. Dichos procesos toman como entrada un bitexto ($\mathcal{B}$) y obtienen una representación equivalente formada por una secuencia de conexiones entre segmentos, de los textos origen ($\mathcal{L}$) y meta ($\mathcal{R}$), identificados como traducción mutua. Estas conexiones representan, por lo tanto, el significado compartido entre los textos que componen el bitexto y se denominan **bipalabras** [MPASM$^+$09].

Definición 6.1 (Bipalabra) *Una bipalabra $b = (s_L, s_R)$ es una entidad única formada a partir de la combinación de dos segmentos de texto $s_L \in \mathcal{L}$ y $s_R \in \mathcal{R}$ que son traducción mutua en $\mathcal{B}$.*

Por lo tanto, la transformación obtenida sobre los procesos de alineamiento textual está formada por una secuencia de bipalabras que mantiene el significado original del bitexto y cuyas propiedades dependen directamente de la estrategia utilizada para la conexión de los segmentos. En el presente trabajo, se consideran diferentes estrategias de alineamiento *uno a uno* y *uno a muchos*. Esto supone que el segmento izquierdo de la bipalabra (s_L) es una única palabra en el texto origen, mientras que el segmento derecho (s_R) está formado por una secuencia de una o más palabras en el texto meta.

Los procesos de alineamiento textual comprenden, genéricamente, dos etapas complementarias que establecen las relaciones de traducción a diferentes niveles de granularidad. Los procesos de *alineamiento a nivel de frase* identifican, en una primera etapa, aquellas frases que son traducciones mutuas en el bitexto. El algoritmo propuesto por Gale y Church [GC93] calcula estos alineamientos sobre un sencillo modelo estadístico basado en la longitud de las frases que componen el bitexto. La propiedad principal de esta propuesta es que las frases de gran tamaño en uno de los idiomas tienden a traducirse en frases de gran tamaño en el idioma complementario. Una segunda etapa de *alineamiento a nivel de palabra* [BPPM93, ON03] se lleva a cabo sobre la anterior, obteniendo una nueva representación en la que aparecen relacionadas aquellas palabras identificadas como traducciones mutuas. La figura 6.1 muestra un ejemplo de alineamiento a nivel de palabra sobre dos frases previamente alineadas en un bitexto castellano-inglés (`es-en`). Este resultado se obtiene al fijar una estrategia de alineamiento "uno a uno" de tal forma que cada palabra en el texto origen se alinea, a lo sumo, con una palabra en el texto meta.

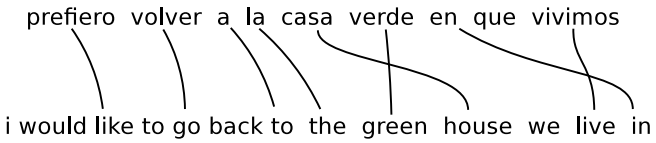


Figura 6.1: Ejemplo de alineamiento "uno a uno" en un bitexto es-en.

Inicialmente, el bitexto se preprocesa para facilitar la identificación de los alineamientos existentes a nivel de palabra. Este paso es común al desarrollado en aquellas aplicaciones de procesamiento de lenguaje natural cuya funcionalidad contempla la utilización de bitextos y, básicamente, consiste en normalizar el bitexto a letras minúsculas. Los alineamientos a nivel de palabra se obtienen con la herramienta de código abierto GIZA++[1] [ON03] que implementa los modelos estadísticos clásicos para este tipo de alineamientos [BPPM93, VNT96]. GIZA++ produce alineamientos que relacionan cada palabra en $\mathcal{L}$ con al menos una palabra en $\mathcal{R}$ mientras que cada palabra en $\mathcal{R}$ puede estar alineada con varias palabras en $\mathcal{L}$. Esto supone un alineamiento "muchos a uno". Intercambiando los roles $\mathcal{L}$ y $\mathcal{R}$ de los textos se obtiene un nuevo alineamiento "uno a muchos", desde la perspectiva del bitexto original. Finalmente, el alineamiento *uno a uno* se obtiene mediante la *intersección* de los dos alineamientos anteriores. Esta transformación final del bitexto es la considerada como punto de partida en el capítulo actual. De

[1] `http://code.google.com/p/giza-pp/`

acuerdo con ello, el ejemplo de alineamiento mostrado en la figura 6.1 se consolida, finalmente, en la siguiente secuencia de bipalabras:

```
(,i) (,would) (prefiero,like) (,to) (volver,go) (,back) (a,to) (la,the)
(,green) (casa,house) (verde,) (,we) (,live) (en,in) (que,) (vivimos,)
```

Nótese que algunas bipalabras, por ejemplo `(,green)`, están formadas por una única palabra. Esto es debido a que el alineamiento obtenido para dicha palabra ha sido descartado al originar un "cruce" en la representación del bitexto. En el ejemplo comentado, el alineamiento entre "`verde`" y "`green`" se descarta dado que cruza sobre el alineamiento establecido entre las palabras "`casa`" y "`house`". El descarte de alineamientos cruzados es necesario para poder reconstruir el bitexto original en la etapa de descompresión. Con el fin de mantener la representación del bitexto como una secuencia de bipalabras, se considera la existencia de una palabra vacía (ϵ) en cada uno de los idiomas de tal forma que en aquellos bipalabras en las que sólo aparece una palabra se considera que ésta se traduce por ϵ, en el texto complementario.

Las consideraciones anteriores permiten obtener transformaciones del bitexto en las que, a pesar de los beneficios obtenidos sobre la relación de traducción representada, se identifican algunas debilidades:

1. La transformación del bitexto sólo considera alineamientos *uno a uno* $(1:1)$ lo que impide aprovechar algunas relaciones de traducción significativas. Por ejemplo, en el caso anterior, la palabra "`vivimos`" se traduce en inglés por "`we live`". Para representar este tipo de relaciones, el esquema actual se debe extender de tal forma que soporte alineamientos *uno a muchos* $(1:N)$.

2. El descarte de alineamientos cruzados impide la representación de las relaciones de traducción que se crucen sobre alguna otra. Esto supone una debilidad importante para el alineamiento de pares de idiomas poco relacionados debido a que un cambio en el orden de las palabras de un idioma, respecto al orden de sus traducciones en el idioma complementario, conlleva el descarte de alineamientos significativos por su cruce sobre algún otro.

La primera de las situaciones se afronta sin más que redefinir la unidad simbólica considerada en la parte derecha del alineamiento, de forma que ésta pueda contener una secuencia de una o más palabras (estos símbolos se denominan *multipalabras*). La formación de estas multipalabras puede considerar secuencias de palabras contiguas y no contiguas en $\mathcal{R}$. Ambos tipos de multipalabras pueden observarse en la figura 6.2 que muestra el resultado obtenido al aplicar una estrategia de alineamiento "*uno a muchos*" sobre el ejemplo previamente comentado. Por un lado, "`to go back`" y "`we live`" (identificadas en el ámbito de las bipalabras `(volver,to go back)` y `(vivimos,we live)`) están formadas por palabras contiguas en el texto meta, mientras que las palabras "`i`" y "`like`" (consideradas en el segmento derecho de la bipalabra `(prefiero,i like)`) no están contiguas en dicho texto.

Por su parte, la resolución del segundo de los problemas planteados se lleva a cabo, sobre la decisión anterior, añadiendo un conjunto de valores de *desplazamiento* en cada bipalabra. Estos valores se interpretan, a la hora de restaurar el texto meta, como el número de palabras que deben ser previamente añadidas al flujo de salida antes de introducir la actual. La configuración de estos conjunto de desplazamientos responde a las siguientes propiedades:

- El conjunto de desplazamientos contiene un único valor en todas aquellas bipalabras cuyo segmento derecho sea una multipalabra formada por una única palabra.

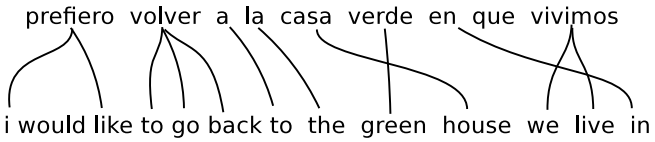


Figura 6.2: Ejemplo de alineamiento "uno a muchos" en un bitexto es-en.

- Al igual que en el caso anterior, el conjunto de desplazamientos está formado por un único elemento en todas aquellas bipalabras cuyo segmento derecho sea una multipalabra formada por una secuencia de palabras contiguas en el texto meta. Este valor, referido a la primera palabra en la secuencia, indica el número de posiciones que la multipalabra se desplaza de acuerdo a su relación con la palabra del texto origen.

- Finalmente, la consideración de multipalabras no contiguas, en el segmento derecho de la bipalabra, requiere un conjunto de desplazamiento con tantos elementos como palabras formen dicha multipalabra. El primero de los valores indica el número de posiciones en las que se desplaza la primera palabra, mientras que los valores restantes representan el desplazamiento de cada una de las subsiguientes palabras respecto a su predecesora en el ámbito de la multipalabra.

Nótese que, para el primer y el segundo caso, el conjunto de desplazamientos se puede descartar en todas aquellas bipalabras en las que el segmento derecho no se encuentre desplazado respecto a la palabra asociada en el texto origen. De acuerdo con estas propiedades, el alineamiento presentado en la figura 6.2 se representa mediante la siguiente secuencia de bipalabras:

```
(prefiero,i like,0,1) (,would) (volver,to go back) (a,to) (la,the)
 (casa,house,1) (verde,green) (en,in,2) (que,) (vivimos, we live)
```

Analizando la primera bipalabra se pueden observar todas las propiedades añadidas con las decisiones anteriores. Por un lado, se establece una relación de traducción entre la palabra "`prefiero`" y la multipalabra "`i like`", cuyas palabras componentes no son contiguas en $\mathcal{R}$. Esta última propiedad implica la necesidad de utilizar dos desplazamientos: el primero de ellos indica que la palabra "`i`" no está desplazada respecto a "`prefiero`", mientras que el segundo muestra la necesidad de incluir una palabra entre "`i`" y "`like`" al regenerar el texto meta.

6.2.1. Terminología

Como se ha planteado anteriormente, un bitexto $\mathcal{B}$ está formado por sendos textos (*origen*: $\mathcal{L}$ y *meta*: $\mathcal{R}$) que contienen la misma información en dos idiomas diferentes. Las diferentes técnicas de modelado, propuestas en la siguiente sección, toman como entrada una representación de los bitextos acorde a las propiedades anteriormente explicadas. Por lo tanto, se asume que estas representaciones son secuencias de bipalabras (ver Definición 6.1) obtenidas de acuerdo a las propiedades de las estrategias de alineamiento *uno a uno* y *uno a muchos* previamente indicadas. Estas estrategias se concretan sobre cuatro políticas específicas con las siguientes características:

`1:1 No Desp.` implementa un alineamiento *uno a uno* sin valores de desplazamiento en las bipalabras. Es la estrategia más sencilla de las cuatro ya que restringe a una única palabra el segmento derecho de la bipalabra y descarta los alineamientos cruzados.

`1:1 Desp.` implementa, al igual que la anterior, un alineamiento *uno a uno*, aunque añade valores de desplazamiento en las bipalabras lo cual permite no descartar alineamientos

cruzados en el bitexto. Al restringirse el segmento derecho a una única palabra, cada bipalabra contiene, a lo sumo, un elemento en su conjunto de desplazamiento.

`1:N Cont.` implementa un alineamiento *uno a muchos* en el que las multipalabras, que forman el segmento derecho, están formadas por palabras contiguas en el texto meta. Al igual que en la política anterior, cada bipalabra contiene un conjunto de desplazamiento formado, como máximo, por un único elemento.

`1:N No Cont.` implementa un alineamiento *uno a muchos* en el que los segmentos derechos pueden contener multipalabras formadas por palabras no contiguas en el texto meta. Esta política es la más compleja de la cuatro, dado que no establece restricciones en cuanto al tamaño máximo de los conjuntos de desplazamiento de cada bipalabra.

De cara a las siguientes secciones, se consideran sendos vocabularios origen y meta asociados, respectivamente, con $\mathcal{L}$ y $\mathcal{R}$. Cada uno de estos vocabularios contiene la palabra vacía ϵ considerada en la formación de bipalabras.

Definición 6.2 (Vocabulario Origen) *El vocabulario origen Σ_L está formado por el conjunto de σ_L palabras diferentes identificadas en $\mathcal{L}$.*

Definición 6.3 (Vocabulario Destino) *El vocabulario meta Σ_R está formado por el conjunto de σ_R multipalabras diferentes identificadas en $\mathcal{R}$.*

La configuración de Σ_R es similar para `1:1 No Desp.` y `1:1 Desp.` ya que el vocabulario está formado, exclusivamente, por las palabras identificadas en $\mathcal{R}$. Para `1:N Cont.` y `1:N No Cont.`, Σ_R contiene tanto las palabras anteriores como las multipalabras contiguas y no contiguas, respectivamente, identificadas en $\mathcal{R}$.

De forma complementaria a las definiciones anteriores, se considera un tercer vocabulario (de bipalabras) directamente asociado con el bitexto $\mathcal{B}$.

Definición 6.4 (Vocabulario de Bipalabras) *El vocabulario de bipalabras Σ_B está formado por el conjunto de σ_B bipalabras $b = (s_L, s_R)$ diferentes identificadas en $\mathcal{B}$. De una manera más formal, $\Sigma_B = \{b = (s_L, s_R)/s_L \in \Sigma_L \wedge s_R \in \Sigma_R\}$.*

Finalmente, el tamaño de un bitexto se estima a partir del número total de bipalabras que lo forman: n. Esto supone que, de acuerdo a las propiedades explicadas para las secuencias de bipalabras, cada uno de los textos en el bitexto está formado por n palabras. Salvo que se especifique lo contrario, el término palabra se utilizar para referir, indistintamente, tanto a las palabras identificadas en $\mathcal{L}$ como a las multipalabras consideradas en $\mathcal{R}$.

6.3 Representaciones de bitextos orientadas a compresión

Un bitexto está formado por dos textos en lenguaje natural con un contenido semántico común. De esta manera, la compresión de bitextos se puede considerar como una disciplina específica en el área de compresión de texto, con la particularidad de que el contenido compartido en el bitexto supone una fuente de redundancia complementaria que debe ser tratada de forma específica. Esta caracterización supone que la base inicial, sobre la que afrontar este problema, debe considerar la experiencia previa adquirida en compresión de lenguaje natural.

Aunque cada idioma posee unas propiedades específicas, todos aquellos con los que vamos a tratar en el presente capítulo mantienen el concepto de *palabra* como unidad mínima de información. Sobre esta premisa, y considerando lo planteado en los dos capítulos anteriores, la

utilización de un modelo orientado a palabras plantea la solución de partida lógica para obtener una representación inicial del bitexto. Sin embargo, estas representaciones deben ser refinadas, respecto a las anteriores, con el fin de adaptarse a las propiedades específicas de este tipo de textos. Más adelante se analizan las diferencias estadísticas de un bitexto respecto a un texto genérico en lenguaje natural, considerando que la principal diferencia no es de tipo estadístico. La *relación de traducción mutua* existente entre los dos textos que componen el bitexto representa la propiedad más importante que se debe mantener sobre el modelo de representación.

Tal y como se comentaba previamente, la secuencia de bipalabras obtenida como resultado de un proceso de alineamiento textual contiene las relaciones de traducción identificadas en el bitexto. Sin embargo, la representación de este "bitexto transformado" sobre un modelo genérico orientado a palabras no garantiza el mantenimiento de la relación de traducción obtenida. La Sección §6.3.1 estudia este problema sobre un *baseline* general a partir del que se obtiene un conjunto de resultados específicos que permiten la representación de un bitexto sobre un modelo orientado a palabras. Esto significa que cada segmento de la bipalabra se gestiona y codifica de forma independiente, de tal forma que la representación de cada bipalabra requiere dos palabras de código independientes. De forma complementaria, la Sección §6.3.2 analiza la posibilidad de obtener una propuesta alternativa capaz de modelar directamente las propiedades básicas de un bitexto. Para ello se utiliza el concepto de bipalabra como unidad simbólica de representación, de manera que cada bipalabra se gestiona y codifica mediante una sola palabra de código.

6.3.1. Representaciones basadas en palabras

Tanto las propuestas que se presentan en este trabajo, como el conjunto de técnicas repasadas en los capítulos previos, demuestran la efectividad de los modelos de palabras para la representación (orientada a compresión) de textos en lenguaje natural. Sin embargo, la compresión específica de bitextos plantea un nuevo reto en este área dado que no sólo requiere el modelado de sus propiedades genéricas como lenguaje natural sino que también precisa representar la relación de traducción mutua existente entre sus textos constituyentes. Desde la perspectiva de la compresión, esta relación facilita la identificación y eliminación de la redundancia semántica existente en el bitexto. Pero, desde el punto de vista de su utilización, la relación de traducción es la que posibilita el uso de los bitextos en las aplicaciones previamente señaladas.

El modelado de la relación de traducción, sobre una propuesta orientada a palabras, requiere que los términos equivalentes en ambos idiomas aparezcan contiguos en el texto comprimido. Esto facilita la localización de la traducción de una determinada palabra con independencia del texto al que pertenece: la traducción de una palabra del texto origen se codifica justo a continuación de ésta, lo que supone que la traducción de una palabra en el texto meta se localiza en la posición anterior a la misma. Este tipo de representación se puede obtener sin más que alternar la codificación de una palabra en el texto origen con su correspondiente traducción en el texto meta. Y eso, a su vez, se consigue de forma sencilla sin más que seguir la secuencia de bipalabras resultante del proceso previo de alineamiento textual.

En primer lugar se definen dos propuestas de representación basadas, exclusivamente, en las propiedades genéricas del lenguaje natural. Por lo tanto, dichas propuestas manejan de forma indirecta la relación de traducción contenida en la secuencia de bipalabras que representa el bitexto. Estas propuestas constituyen un *baseline* general dado que no son capaces de detectar y eliminar la redundancia semántica inherente al bitexto y, por tanto, las tasas de compresión esperadas para ellas serán menos competitivas que las obtenidas por las técnicas específicamente planteadas para la compresión de bitextos (ver Sección §6.4). Sin embargo, estas propuestas sí facilitan la obtención de una representación compacta del bitexto sobre la que se puede hacer uso de la relación de traducción contenida en la secuencia de bipalabras.

Baseline. El *baseline* actual se constituye sobre sendas propuestas basadas en diccionario y orientadas a palabra. Como se planteaba previamente, esto supone manejar y codificar cada palabra en la bipalabra como un símbolo independiente. De forma genérica, las técnicas de diccionario (ver Sección §3.5) se centran en la identificación y organización del vocabulario de símbolos que componen un determinado texto. Esto supone, para el caso actual, gestionar los símbolos procedentes de dos vocabularios diferentes: Σ_L y Σ_R. Esta propiedad plantea una notable diferencia respecto a las distribuciones de palabras encontradas en un texto en lenguaje natural. Por tanto, las propuestas actuales se centran en encontrar una representación del diccionario que minimice el tamaño del bitexto comprimido.

La solución más sencilla consiste en utilizar un *vocabulario único* (`1V`) para la representación conjunta de Σ_L y Σ_R. Esto implica la necesidad de manejar, por duplicado, la palabra vacía ϵ contenida en ambos vocabularios. Esta solución es equivalente a las utilizadas para la compresión de lenguaje natural utilizando códigos densos (ver §3.3.3). La propuesta `1V` utiliza una sóla función de diccionario para el mapeo del vocabulario: $\mathcal{D}_{1v} = \{\Sigma_L \cup \Sigma_R \rightarrow \mathbb{N}\}$. El número de palabras en $\mathcal{D}_{1v}$ es $\sigma_L + \sigma_R$. Esto implica que el tamaño del vocabulario depende directamente de la política de alineamiento utilizada y, por tanto, del valor σ_R.

La *Ley de Heaps* [Hea78, BYRN99] (ver Sección §3.1) predice que el número de palabras diferentes es $O(n^\beta)$ para un texto formado por n palabras totales. Esto supone un crecimiento sublineal del tamaño del vocabulario respecto al tamaño del texto. Para el caso de un bitexto, la Ley de Heaps podría ser aplicada de forma independiente a cada uno de sus textos componentes considerando que la configuración final de la función $\mathcal{D}_{1v}$ debe ser capaz de integrar las propiedades particulares de Σ_L y Σ_R.

Supongamos un bitexto formado por formado por n bipalabras totales. De acuerdo a lo establecido anteriormente, se conoce que tanto $\mathcal{L}$ como $\mathcal{R}$ contienen un total de n palabras. El tamaño del vocabulario de un bitexto es, de acuerdo con la Ley de Heaps, $O(n^{\beta_L} + n^{\beta_R})$, donde β_L es un parámetro específico del idioma representado en $\mathcal{L}$ y β_R es, a su vez, un parámetro dependiente del idioma representado en $\mathcal{R}$ y de la política de alineamiento utilizada para la generación de bipalabras. En términos prácticos, el tamaño del vocabulario obtenido en `1V` es superior al requerido para un texto generérico en lenguaje natural compuesto compuesto por un total de $2n$ palabras. Esta propiedad se estudia en la Sección §6.5.1.

Por otra parte, la distribución de palabras que conforman un bitexto también varía respecto a la obtenida sobre un texto genérico en lenguaje natural. La *Ley de Zipf* (ver §3.1) aproxima la naturaleza asimétrica que define la distribución de frecuencia obtenida para las palabras identificadas en un texto: por un lado, una pequeña minoría posee una alta frecuencia de aparición mientras que el conjunto de palabras restante está caracterizado por frecuencias muy bajas. Dicha Ley de Zipf estima la frecuencia relativa de la i-ésima palabra más frecuente como A_x/i^θ, donde A_x es un factor de normalización y θ es un valor dependiente del texto modelado (por ejemplo, $1 < \theta < 2$ para textos en inglés). Sin embargo, la distribución de palabras que conforman un bitexto viene caracterizada por la combinación de las distribuciones de palabras que definen $\mathcal{L}$ y $\mathcal{R}$. De acuerdo a la Ley de Zipf, la i-ésima palabra más frecuente en $\mathcal{L}$ posee un frecuencia de aparición $1/i^{\theta_L}$ mientras que su equivalente en $\mathcal{R}$ se caracteriza por una frecuencia $1/i^{\theta_R}$. Por lo tanto, la función de diccionario $\mathcal{D}_{1v}$ depende de los valores θ_L y θ_R:

- Si $|\theta_L - \theta_R| \approx 0$ la frecuencia de la i-ésima palabra más frecuente en cada texto tiende a coincidir: $1/i^{\theta_L} \approx 1/i^{\theta_R}$. Esta propiedad hace más uniforme la distribución de frecuencias de las palabras del bitexto. Esto conlleva un aumento en la longitud media de la palabra de código utilizada para la codificación del bitexto.

- Si, por el contrario, $\theta_L > \theta_R$ (equivalentemente para $\theta_L < \theta_R$) la distribución de frecuencia de las palabras de $\mathcal{L}$ está por debajo de la $\mathcal{R}$. Esta propiedad penaliza la significatividad

relativa de las palabras de $\mathcal{L}$ dentro del bitexto de tal forma que serán necesarias palabras de código de mayor longitud a las requeridas si el texto se codificase independientemente.

Las propiedades anteriores sugieren que el modelado y codificación de $\mathcal{L}$ y $\mathcal{R}$, desde una perspectiva orientada a palabras, debe llevarse a cabo de forma independiente para cada uno de los textos aprovechando sus propiedades específicas como lenguaje natural. Esta premisa es la que guía el desarrollo de la segunda técnica considerada en este *baseline*: `2V`.

`2V` afronta las debilidades de `1V` sobre su misma filosofía. Esto supone mantener la orientación a palabra para la representación del bitexto y utilizar una función de diccionario para su codificación. Sin embargo, `2V` considera la representación independiente de los vocabularios Σ_L y Σ_R, lo que implica la necesidad de definir una función de diccionario específica para cada uno de estos vocabularios: $\mathcal{D}^L_{2v} = \{\Sigma_L \rightarrow \mathbb{N}\}$ y $\mathcal{D}^R_{2v} = \{\Sigma_R \rightarrow \mathbb{N}\}$. Esta decisión subsana las debilidades encontradas en `1V` al plantear una representación independiente de cada texto en el bitexto. La alternancia en la codificación entre palabras de $\mathcal{L}$ y $\mathcal{R}$ mantiene adyacentes, en el bitexto comprimido, cada palabra y su traducción.

Desde el punto de vista espacial, tanto `1V` como `2V` tienen un coste $O(\sigma_L + \sigma_R)$. Sin embargo, la representación de $\mathcal{L}$ y $\mathcal{R}$ mediante dos funciones de diccionario independientes reduce notablemente el tamaño del alfabeto de entrada al codificador. La representación de un bitexto genérico sobre `1V` genera flujos de enteros distribuidos en el rango $[1, \sigma_L + \sigma_R]$ mientras que el rango de enteros requerido por `2V`, para el mismo bitexto, se distribuye en $[1, max(\sigma_L, \sigma_R)]$. Esta evolución provoca, a su vez, el aumento de la frecuencia de utilización de los $min(\sigma_L, \sigma_R)$ primeros enteros, favoreciendo la utilización de palabras de código más cortas para su codificación. Supongamos un bitexto *castellano-inglés* (`es-en`) dentro del que se identifican las frases "*la casa verde en que vivimos*" y "*the green house we live in*". El alineamiento del bitexto original, con la política `1:1 No Desp.`, obtiene la siguiente secuencia de bipalabras:

```
(la,the) (,green) (casa,house) (verde,) (,we) (,live) (en,in) (que,) (vivimos,)
```

La figura 6.3 muestra las funciones de diccionario obtenidas con `1V` y `2V` para la representación del fragmento de bitexto anterior. En la parte inferior de la figura se muestran la secuencias de enteros utilizadas como base para la codificación del bitexto sobre cada una de las propuestas. Puede observarse como el tamaño del alfabeto de entrada al codificador es menor para `2V` cuya distribución de enteros muestra una mayor frecuencia para los primeros valores. Esta propiedad permite mejorar la efectividad de `2V` frente a `1V` al utilizar un código de enteros para la compresión de la secuencias.

Modelado de orden 1 sobre relación de traducción. La descripción dada por Melamed [Mel01], acerca de la naturaleza de un bitexto, plantea la idea básica sobre la que se desarrolla la propuesta de modelado actual. Si observamos $\mathcal{R}$ como una anotación detallada de $\mathcal{L}$, entonces podemos utilizar $\mathcal{L}$ como contexto de representación para $\mathcal{R}$. Esto supone, básicamente, utilizar las palabras identificadas en Σ_L como contexto para la representación de las palabras de Σ_R con las que se relacionan de acuerdo a la secuencia de bipalabras obtenidas tras el alineamiento del bitexto original.

La efectividad de esta propuesta se basa en el hecho de que una palabra tiene un pequeño número de traducciones en el idioma complementario. Este conjunto de traducciones conforma el *vocabulario de traducción* de una palabra en el texto origen:

Definición 6.5 (Vocabulario de Traducción) *Dada una palabra $p_L \in \Sigma_L$, se define su vocabulario de traducción $\tau(p_L) = \{p_R \in \Sigma_R / (p_L, p_R) \in \Sigma_B\}$.*

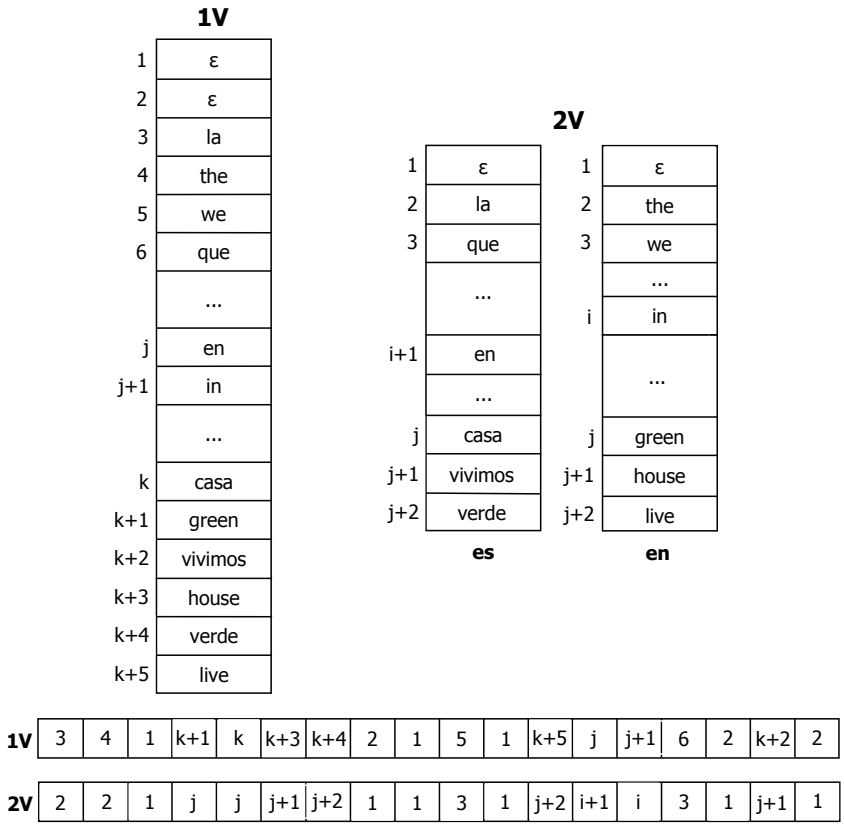


Figura 6.3: Representación y codificación de un bitexto con 1V y 2V.

De esta manera, cada palabra en $p_R \in \Sigma_R$ pasa a ser representada en los vocabularios de traducción de todas aquellas palabras $p_L \in \Sigma_L$ con las que se relaciona de acuerdo a las diferentes bipalabras en las que interviene. La probabilidad de ocurrencia de p_R en cada $\tau(p_L)$ se calcula a partir del número de veces que la (p_L, p_R) aparece en la secuencia de bipalabras.

La necesidad de considerar ϵ en Σ_L requiere un vocabulario de traducción $\tau(\epsilon)$ de gran tamaño en el que se almacenan todas aquellas palabras en σ_R que no han podido ser alineadas en alguna de sus ocurrencias en el bitexto. Esto supone, en el peor caso, codificar cada palabra en $\tau(\epsilon)$ con un modelo de orden 0 sobre Σ_R.

Por su parte, las apariciones de ϵ en $\mathcal{R}$ se modelan de forma similar al resto de palabras en σ_R. Esto supone su representación en los vocabularios $\tau(p_L)$ en los que se almacena de acuerdo a las bipalabras (p_l, ϵ) identificadas en $\mathcal{B}$. Su impacto se puede estimar, empíricamente, analizando la distribución de bipalabras (p_l, ϵ) esperada en el bitexto:

1. Estas bipalabras se utilizan en aquellos casos en los que la relación de traducción de la palabra p_L no ha podido ser establecida. Estas situaciones tienden a ser poco frecuentes, por lo que la probabilidad de ocurrencia ϵ en $\tau(p_L)$ tiende a ser baja y, por lo tanto, no perjudica la codificación del resto de palabras en $\tau(p_L)$ cuya probabilidad tiende a ser mayor al ser "traducciones válidas" de p_L.

2. Estas bipalabras se utilizan, complementariamente, para la representación de alineamientos descartados por su cruce sobre otros alineamientos. Este tipo de situación puede ser más frecuente que la anterior, dependiendo del par de idiomas contenido en el bitexto. Esto supone una mayor probabilidad de ocurrencia de ϵ en $\tau(p_L)$, promocionando su codificación en dicho contexto.

Utilizaremos la figura 6.1 para estudiar con mayor detenimiento este segundo caso: el adjetivo *verde* no está alineado con ninguna palabra en el texto inglés. Este es un hecho bastante común en el par de idiomas castellano-inglés considerado en este bitexto, dado que el orden que siguen las partes componentes de las frases no coincide en ambos idiomas. Por lo tanto, es muy probable que los adjetivos en castellano tiendan a no estar alineados con su traducción en inglés y bipalabras como (`verde`,ϵ) tengan una alta frecuencia de aparición en $\mathcal{B}$. De esta manera, la probabilidad de ocurrencia de ϵ en vocabularios como $\tau(verde)$ tiende a ser alta, lo cual favorecerá su posterior codificación. Al contrario que en el caso 1, en estas situaciones se pretende favorecer la codificación de ϵ en vocabularios como $\tau(verde)$ dado que este tipo de regularidades son frecuentes entre ciertos pares de idiomas y el modelo debe ser capaz de gestionarlas de cara a su codificación efectiva. De forma ortogonal a este razonamiento, se debe considerar que las políticas de alineamiento que utilizan desplazamientos en la bipalabra tienden a minimizar la ocurrencia del fenómeno anterior. Esto tienden a caracterizar este problema sobre las propiedades definidas en el caso 1 y, por tanto, a no promocionar la codificación de ϵ en aquellos $\tau(\epsilon)$ en los que presente una baja probabilidad de ocurrencia.

La propuesta actual guarda ciertas similitudes con el trabajo de Nevill-Manning y Bell [NMB92]. Por un lado, modelamos el texto $\mathcal{L}$ de acuerdo a sus propiedades particulares como lenguaje natural y con independencia de la relación de traducción inherente al bitexto (excepto por la necesidad de considerar ϵ en Σ_L). Esto otorga una gran flexibilidad al modelado actual dado que permite utilizar técnicas genéricas para lenguaje natural cuya elección dependerá del contexto en el que vaya a ser utilizada. Por otro lado, se plantea la representación de $\mathcal{R}$ a partir del modelo de orden 1 obtenido a partir de la relación de traducción derivada de la secuencia de bipalabras. La principal diferencia respecto al trabajo de Nevill-Manning y Bell está en este punto. La representación del bitexto que obtenemos a través de la secuencia de bipalabras nos permite obtener una información más precisa de la relación de traducción inherente al bitexto y con ello aumentar la efectividad de las técnicas de compresión construidas sobre este tipo de modelo. Esta información se almacena en los vocabularios de traducción $\tau(p_L)$ asociados a cada palabra $p_L \in \Sigma_L$ y se pone a disposición del codificador a través de funciones de diccionario específicas para cada uno de estos vocabularios: $\mathcal{D}_{p_L} = \{p_R \rightarrow \mathbb{N}/(p_L, p_R) \in \Sigma_B\}$. Estas funciones asignan un identificador local a cada una de las palabras de Σ_R almacenadas en cada vocabulario $\tau(p_L)$.

6.3.2. Representaciones basadas en bipalabras

La propuesta de representación actual mantiene que la palabra es la unidad mínima de significado en cada uno de los textos, por lo cual la bipalabra contiene la menor relación de traducción que puede ser identificada en el bitexto. Esto se materializa en un modelo de representación que interpreta cada bipalabra como un símbolo único. Esto es, las dos palabras que forman la bipalabra se modelan y codifican bajo la misma entidad. De esta manera, las representaciones basadas en bipalabras construyen un modelo del bitexto basado en el significado compartido por los textos que lo forman.

Este planteamiento permite aprovechar las propiedades del bitexto como lenguaje natural y, a su vez, sumar las propiedades de significado compartido inherente al propio bitexto.Por un lado, el modelo actual mantiene una representación basada en palabras para cada uno de los textos dado que la secuencia de bipalabras se puede observar, al igual que en las propuestas anteriores, como una representación en alternancia de palabras en $\mathcal{L}$ y $\mathcal{R}$. Por otro lado, manejar cada bipalabra como una entidad única plantea una representación orientada al concepto representado por el significado compartido por las palabras que componen la bipalabra.

Los modelos basados en bipalabras obtienen una representación más compacta del bitexto al reducir el número de símbolos totales necesarios para su representación. Nótese que mientras

las propuestas anteriores procesan n palabras en $\mathcal{L}$ y otras n palabras en $\mathcal{R}$ (un total de $2n$ símbolos), la propuesta actual requiere un total de n símbolos para representar y codificar cada una de las n bipalabras en $\mathcal{B}$. Sin embargo, el modelado de la bipalabra como un símbolo único también supone ciertas desventajas en la representación obtenida.

La representación de relaciones de *sinonimia* y *polisemia* conlleva la necesidad de utilizar un número variable de bipalabras para representar un mismo concepto o para la representación de una misma palabra utilizada en diferentes conceptos. Supongamos un bitexto `es-en` y, por ejemplo, la palabra castellana "*caña*". Según el diccionario de la Real Academia Española[2], esta palabra tiene hasta veintiuna acepciones diferentes. Esta propiedad implica que una única palabra podría representar hasta veintiún conceptos diferentes lo que, a su vez, significa un número de bipalabras variable y dependiente de la posible relación de sinonimia/polisemia existente en el idioma complementario. Seleccionaremos algunas acepciones de esta palabra para entender el impacto de estas relaciones semánticas en las representaciones basadas en bipalabras:

- `caña`: *tallo de las plantas gramíneas, por lo común hueco y nudoso.* La palabra `reed` podría ser utilizada como traducción en el texto inglés. Esto, en el mejor caso, añade una única bipalabra a Σ_B: `(caña, reed)`.

- `caña`: *líquido contenido en este vaso* (relativo a la cerveza). La palabra `lager` podría considerarse una buena traducción en este caso. Sin embargo consideremos la posibilidad de que ante una ocurrencia de la palabra `lager`, el texto castellano considere directamente la palabra `cerveza` como traducción. En este mismo contexto, la palabra `cerveza` se traduce como `beer`. En conclusión a partir de una de las acepciones de `caña`, se añadirían a Σ_B las bipalabras; `(caña, lager)`, `(cerveza, lager)` y `(cerveza, beer)`.

- `caña`: *de pescar*. Para esta traducción se considera la palabra `rod` que, a su vez, podría ser traducida al castellano como `vara` o `barra`. De la misma forma, palabras como `stick` o `bar`, entre otras, podrían considerarse traducciones de las anteriores. Esto supone la necesidad de añadir a Σ_B las siguientes bipalabras: `(caña, rod)`, `(vara, rod)`, `(barra, rod)`, `(vara, stick)` y `(barra, bar)`.

Estos ejemplos demuestran como las relaciones de sinonimia y polisemia existentes en lenguaje natural suponen un importante crecimiento en el valor de σ_B. Además, se debe considerar que cada una de las palabras existentes en el ejemplo actual podría no estar alineada en alguna de sus apariciones, lo que conlleva la necesidad de representar nuevas bipalabras de la forma `(caña,` ϵ`)`, `(`ϵ`, reed)`, `(cerveza,` ϵ`)`, `(`ϵ`, lager)`, etc. Esta última propiedad depende del par de idiomas representados en el bitexto (para aquellos pares menos relacionados, este conjunto de alineamientos con ϵ puede suponer una fracción importante del tamaño del Σ_B). En la sección de experimentación se plantea un estudio estadístico de este conjunto de propiedades y el impacto que suponen en la representación comprimida del bitexto.

Los modelos utilizados para las representaciones basadas en bipalabras consideran las propiedades de este símbolo a la hora de obtener un almacenamiento eficiente para su vocabulario. Como se planteaba anteriormente, el concepto de bipalabra se desarrolla a partir de las palabras identificadas en cada uno de los textos. Esto sugiere almacenar por un lado los vocabularios Σ_L y Σ_R y, finalmente, construir Σ_B a partir de los anteriores. Para ello se considera una *jerarquía en dos niveles*.

Nivel 1: contiene la representación de los vocabularios Σ_L y Σ_R. Esto supone gestionar de forma independiente la distribución de palabras existente en $\mathcal{L}$ y $\mathcal{R}$ y con ello aprovechar

[2]Disponible en `http://buscon.rae.es/draeI/`

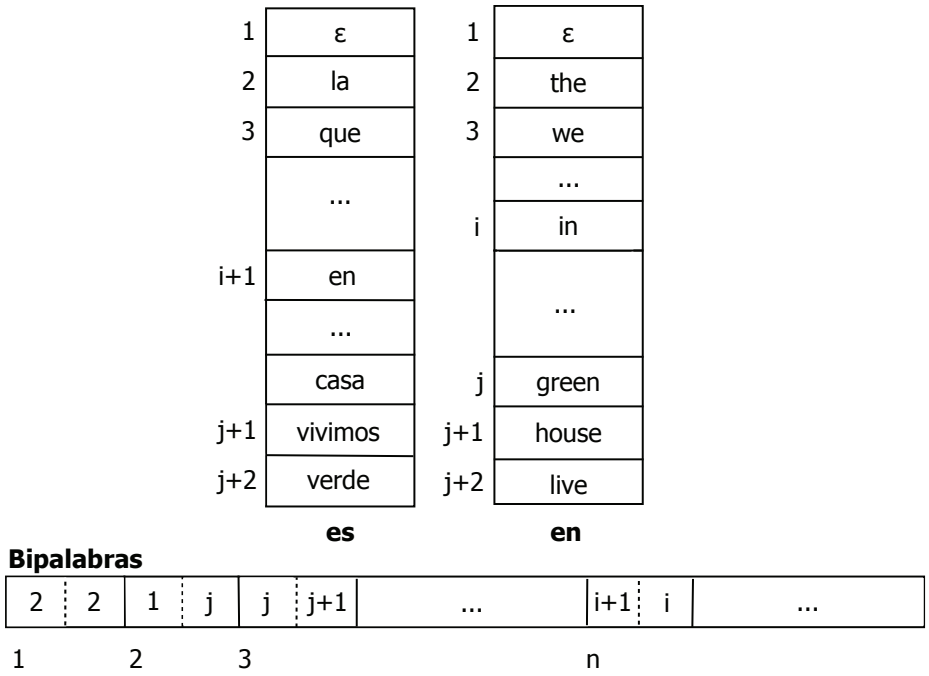


Figura 6.4: Modelo jerárquico para representaciones basadas en bipalabras.

sus propiedades específicas como textos en lenguaje natural. La configuración de este primer nivel es comparable a la propuesta en `2V`.

Nivel 2: almacena Σ_B de tal forma que cada elemento representa un símbolo único (bipalabra) definido a partir de sendas referencias a la representación de sus palabras constituyentes en Σ_L y Σ_R.

La organización de cada uno de los vocabularios y el almacenamiento de las bipalabras en el vocabulario de segundo nivel representan los problemas abiertos para el desarrollo de un compresor sobre este modelado. En la siguiente sección se detalla el compresor `2LCAB` y se explica como implementa cada una de estas cuestiones considerando la definición de una función de diccionario $\mathcal{D}_B = \{\Sigma_B \rightarrow \mathbb{N}\}$ que mapea cada bipalabra en Σ_B en un valor único utilizado como base para su codificación.

La figura 6.4 muestra, sobre el ejemplo planteado anteriormente, la distribución de los vocabularios necesarios en el modelo actual. En el primer nivel se pueden observar los vocabularios Σ_L y Σ_R (referidos, respectivamente, como `es` y `en` de acuerdo al idioma que representan). Nótese como, para el ejemplo actual, la configuración de estos vocabularios coincide con la planteada para la representación del mismo bitexto con `2V` (ver figura 6.3). En la parte inferior de dicha figura 6.4 se muestra una posible configuración de Σ_B. En la posición 1 se almacena la bipalabra `(la, the)` (a través de las referencias a las posiciones 2 de los respectivos vocabularios Σ_L y Σ_R en los que se almacenan las palabras "`la`" y "`the`"); en la posición 2 de Σ_B se almacena la bipalabra `(`ϵ`, green)` mediante referencias a las posiciones 1 y j de Σ_L y Σ_R en las que se almacenan, respectivamente, las palabras "ϵ" y "`green`"; en la posición n se almacena `(en, in)` de acuerdo a las propiedades anteriores y así para el resto de elementos en el vocabulario. Cada bipalabra en Σ_B se identifica, de acuerdo a la función $\mathcal{D}_B$, por su posición en el vocabulario. Por ejemplo, la bipalabra `(la, the)` estaría identificada, en el ejemplo anterior, por el valor 1.

6.4 Compresión de bitextos

Las técnicas de compresión planteadas en la sección actual se desarrollan sobre las propuestas de modelado presentadas en la sección anterior. En primer lugar se tratan dos problemas independientes del tipo de modelado seleccionado para el compresor. En la Sección §6.2 se planteaban diferentes políticas de alineamiento sobre las que obtener la secuencia de bipalabras que representa un bitexto. Por un lado se consideraba la posibilidad de utilizar una política de alineamientos *uno a muchos* $(1:N)$ de tal forma que la parte derecha de la bipalabra esté formada por una secuencia de N palabras no necesariamente contiguas en $\mathcal{R}$. El almacenamiento de estas secuencias requiere una política específica que permita aprovechar el hecho de que una palabra puede aparecer en más de una multipalabra diferente. En la Sección §6.4.1 se plantea un modelo jerárquico para la representación de estas multipalabras. A continuación se trata la representación de los conjuntos de desplazamiento considerados para evitar el descarte de alineamientos cruzados. Estos desplazamientos se utilizan tanto en la política $1:1$ como en la $1:N$, con propiedades específicas en cada una de ellas y que, por tanto, requieren sendas políticas para su representación (ver Sección §6.4.2).

Una vez caracterizadas las partes comunes a todas las propuestas de compresión, se presenta cada una de ellas de acuerdo al tipo de modelo que utilizan para la representación de la secuencia de bipalabras. En la Sección §6.4.3 se detallan las propuestas orientadas a palabras considerando, también, la implementación del *baseline* utilizado como base para propósitos de comparación. Finalmente, la Sección §6.4.4 presenta el compresor `2LCAB`, dentro del cual se tratan, de forma específica, todas las posibilidades de operación que éste presenta sobre el bitexto comprimido.

6.4.1. Almacenamiento jerárquico de las multipalabras

La utilización de alineamientos $1:N$ permite aprovechar relaciones de traducción significativas entre una palabra en $\mathcal{L}$ y una secuencia de palabras (no necesariamente contiguas) en $\mathcal{R}$. Esta decisión permite obtener una mejor representación del bitexto considerando que la traducción de una palabra no tiene porqué corresponder con una palabra única en el idioma complementario. En el ejemplo mostrado en la Sección §6.2 puede observarse como "`prefiero`" se traduce por "`i like`" o "`volver`" por "`to go back`". Esta política de alineamiento también permite reducir el número de casos en los que interviene la palabra ϵ (para los ejemplos referidos, puede observarse como los pares `(ϵ,i)`, `(ϵ, to)` y `(ϵ,back)` ya no serían necesarios) obteniendo una reducción en el número total de símbolos necesarios para la representación del bitexto.

La utilización de multipalabras provoca un crecimiento en el valor de σ_R respecto al obtenido para una política de alineamiento *uno a uno*. Sin embargo, el conjunto primitivo de palabras con el que se obtienen las multipalabras es similar para todas las políticas de alineamiento. Esto supone que una palabra puede aparecer en varias multipalabras diferentes y, a su vez, algunas de estas multipalabras pueden formar parte de otras multipalabras de mayor longitud. Para afrontar esta situación se plantea el diseño de una jerarquía binaria de derivación en la que cada multipalabra se representa a partir de la combinación de dos elementos constituyentes.

Supongamos tres palabras "`a`","`b`","`c`" $\in \Sigma_R$, tal que cada una de ellas se identifica como $C_a, C_b, C_c \in \mathbb{N}$. El caso primitivo lo representa una multipalabra compuesta por dos palabras: supongamos que "`a b`" $\in \Sigma_R$ y que se identifica como $\mathtt{C}_{ab} \in \mathbb{N}$. La representación de esta multipalabra en la jerarquía binaria es: $\mathtt{C}_{ab} \rightarrow \mathtt{C}_a\mathtt{C}_b$. Extendamos el razonamiento para tres palabras: "`a b c`" $\in \Sigma_R$, identificada como $\mathtt{C}_{abc} \in \mathbb{N}$. La multipalabra podría representarse en la jerarquía binaria sobre dos combinaciones diferentes: "`a b`" y la palabra "`c`" ($\mathtt{C}_{abc} \rightarrow \mathtt{C}_{ab}\mathtt{C}_c$), o la palabra "`a`" y la multipalabra "`b c`" ($\mathtt{C}_{abc} \rightarrow \mathtt{C}_a\mathtt{C}_{bc}$). La decisión de que opción elegir depende de la caracterización de cada una de ellas.

Como se planteaba en la Sección §3.5.2, la elección de la gramática mínima necesaria para la representación de un conjunto de símbolos es un problema $\mathcal{NP}$-difícil. Para el caso actual se plantea una sencilla heurística que considera el número de multipalabras en el que interviene cada uno de los símbolos que forman la multipalabra a representar. Esto supone, en el ejemplo anterior, que la codificación de "`a b c`" se lleva a cabo considerando, por un lado, el número de multipalabras en las que intervienen "`a b`" y "`c`" y, por otro lado, el número de multipalabras en las que aparecen "`a`" y "`b c`". Esta información está disponible en cada una de las propuestas de modelado planteadas en la sección anterior, de tal forma que la jerarquía de derivación se construye como paso previo al inicio de la etapa de codificación.

Cada multipalabra se codifica mediante la elección del par de símbolos que intervienen en un mayor número de bipalabras. Supongamos que la función `multi(`c_i`)` devuelve el número de multipalabras en los que puede participar un determinado símbolo $c_i \in \Sigma_R$. La codificación de "`a b c`" se llevará a cabo como $\mathtt{C}_{ab}\mathtt{C}_c$ *sii* `multi(a b)+multi(c)` $\geq$ `multi(a)+multi(b c)`. En el caso contrario, la multipalabra se codificaría combinando $\mathtt{C}_a\mathtt{C}_{bc}$, asumiendo que $\mathtt{C}_{bc} \in \mathbb{N}$ es el identificador asociado a la multipalabra "`b c`".

Esta decisión supone que la representación de Σ_R (extendido con multipalabras) codifica el flujo de palabras identificadas en $\mathcal{R}$ junto con la secuencia de enteros que representa la forma del árbol binario de derivación obtenido de acuerdo a las propiedades anteriores. La codificación de este flujo de enteros (distribuido en $[0, \sigma_R]$) se materializa sobre un código de redundancia mínima orientado a bit mientras que la codificación de la secuencia de palabras depende de la técnica implementada.

6.4.2. Codificación de los desplazamientos

El uso de desplazamientos permite evitar el descarte de alineamientos cruzados en el ámbito de una frase. Como se explicaba al principio del capítulo, este valor indica el número de posiciones que está desplazada la palabra actual, en $\mathcal{R}$, respecto a la posición esperada en $\mathcal{L}$ de acuerdo a la traducción dada por la bipalabra.

Los esquemas de representación considerados para los desplazamientos dependen directamente de la política de alineamiento utilizada. El caso más sencillo se da para $1:1$ en el que el conjunto de desplazamiento contiene, a lo sumo, un único valor entero. El número total de desplazamientos (d), cuyo valor es mayor que 0, depende del par de idiomas considerado en el bitexto. Esto supone que la distribución de los valores que conforma la secuencia de desplazamientos está formada por subsecuencias alternativas de ceros y valores positivos que representan el desplazamiento asociado a cada bipalabra. La longitud media de estas subsecuencias depende de la proximidad existente entre los pares de idiomas que conforman el bitexto.

La alternativa seleccionada se centra en la codificación exclusiva de aquellas bipalabras cuyo valor de desplazamiento sea mayor que 0. Para este propósito se considera el uso dos enteros por desplazamiento: 1) para la representación del propio valor de desplazamiento: d_i; y 2) para la representación de su posición en el bitexto (p_i). Para este segundo valor se utiliza una técnica diferencial. Esto supone representar cada posición como su diferencia respecto a la posición anterior en la que se identificó un desplazamiento: $p_i - p_{i-1}$. Para la codificación de estos pares de enteros se utiliza una representación de redundancia mínima obtenida sobre sendos códigos canónicos de `Huffman` [SK64, MT97].

La distribución de desplazamientos para $1:N$ depende de la naturaleza de las multipalabras consideradas. Si la formación de multipalabras se restringe a palabras contiguas en $\mathcal{R}$, los valores de desplazamiento obtenidos se distribuyen de forma similar al caso anterior dado que que d_i se calcula en referencia al primer elemento de la multipalabra. Por lo tanto, la codificación de los desplazamientos mantiene las mismas propiedades planteadas para el caso anterior.

La naturaleza de la distribución de desplazamientos asociada a las multipalabras no contiguas varía respecto a las anteriores. En este caso se utilizan k enteros que referencian el desplazamiento de cada una de las k palabras que forman la multipalabra. El primer entero indica el desplazamiento de la primera palabra sobre las mismas propiedades anteriormente explicadas. Mientras tanto, el segundo y subsiguientes valores indican el desplazamiento de la i-ésima palabra respecto a la palabra que la precede en el ámbito de la multipalabra. El esquema confeccionado para la codificación de estos desplazamientos varía ligeramente respecto al anterior. La representación del desplazamiento asociado a la primera palabra requiere dos enteros que indican, al igual que en los casos anteriores, el número de posiciones que se desplaza respecto a $\mathcal{L}$ y su posición en $\mathcal{R}$. La codificación de estos valores se mantiene sobre el esquema anterior. Sin embargo, representar el desplazamiento de la segunda y subsiguientes palabras en la multipalabra requiere un único valor que indica el número de posiciones que ésta se desplaza en $\mathcal{R}$ respecto a la palabra justo anterior. Por lo tanto, este valor representa una codificación diferencial de la palabra respecto a su predecesora en el ámbito de la multipalabra y se representa utilizando el mismo árbol de `Huffman` considerado para la secuencia de enteros que contiene las posiciones de los desplazamientos en $\mathcal{R}$.

Supongamos el alineamiento `(prefiero,i like,0,1)`. El valor 0 indica que '`i`' no está desplazada respecto a '`prefiero`' mientras que '`like`' lo está en 1 posición respecto a '`i`'. Supongamos que '`i`' se localiza en la posición p_i del bitexto. De acuerdo a la técnica planteada, este desplazamiento se representaría como $(0, p_i - p_{i-1}, 1)$, donde 0 representa el desplazamiento de la primera palabra en la multipalabra respecto a $\mathcal{L}$, $p_i - p_{i-1}$ indica la posición, en $\mathcal{R}$, de este desplazamiento y 1 indica que la segunda palabra en la multipalabra está desplazada una posición respecto a la primera.

6.4.3. Compresores orientados a palabras

Las propuestas de compresión orientadas a palabras toman como punto de partida el *baseline* establecido en la sección anterior. A pesar de las diferencias existentes entre las propuestas `1V` y `2V`, su implementación como técnicas de compresión se lleva a cabo de forma similar.

En términos globales, ambas propuestas contemplan dos pasadas sobre el bitexto. En la primera se identifican los vocabularios Σ_L y Σ_R y se construyen las funciones de diccionario correspondientes de acuerdo a la frecuencia de aparición de cada palabra en el bitexto. `1V` lleva a cabo un proceso global sobre Σ_L y Σ_R que da como resultado la ordenación por frecuencia de las $\sigma_L + \sigma_R$ palabras identificadas en el bitexto. Por su parte, la ordenación en `2V` se desarrolla de forma independiente para Σ_L y Σ_R. En ambos casos, se obtienen las funciones de diccionario comentadas en la sección anterior que proyectan cada palabra sobre un identificador entero único. El resultado de esta primera pasada es una tabla *hash* de pares que asocian cada palabra con su identificador correspondiente (para la implementación de esta estructura se utiliza la propuesta de Justin Zobel[3] ya comentada en los capítulos previos).

La etapa de codificación se lleva a cabo de forma sencilla mediante una segunda pasada sobre el bitexto. Este proceso alterna la identificación y codificación de una palabra en $\mathcal{L}$ con el mismo proceso en $\mathcal{R}$. Esto supone leer la palabra, buscarla en la tabla hash correspondiente (considérese que `1V` maneja una única tabla mientras que `2V` posee una para cada idioma) y codificarla de acuerdo a su identificador mediante una función tipo `encode()` que emite la palabra correspondiente de acuerdo al esquema de codificación seleccionado. En la siguiente sección se evalúa la utilización de codificaciones orientadas a bit y a byte (`Huffman` y `ETDC`, respectivamente) con el fin de analizar diferentes propiedades de las técnicas. El proceso de codificación es similar en todos los casos y requiere un tiempo $O(n)$.

[3] http://www.cs.mu.oz.au/~jz/resources/index.html

Algoritmo 6.1 Compresión TRC

```
1: Σ_L ← ∅;
2: Σ_R ← ∅;
3:
4: for all b ∈ B do
5:    if b.origen ∉ Σ_L then
6:       τ(b.origen) ← ∅;
7:    end if
8:    actualizar_L(b.origen);
9:    actualizar_R(b);
10: end for
11:
12: for all p_L ∈ Σ_L do
13:    procesar_τ(p_L);
14: end for
15:
16: for all b ∈ B do
17:    codificar_L(b.origen);
18:    codificar_R(b);
19:    codificar_desplazamiento(b.desp);
20: end for
```

Algoritmo 6.2 Procedimiento actualizar_$\mathcal{R}$

```
Require: b;
1: if b.meta ∉ Σ_R then
2:    Σ_R ← Σ_R ∪ {b.meta}
3: end if
4:
5: if b.meta ∉ τ(b.origen) then
6:    τ(b.origen) ← τ(b.origen) ∪ {b.meta}
7: end if
8: actualizar(τ(b.origen), b.meta);
```

Translation Relationship-based Compressor (`TRC`). Las técnicas `TRC` se definen bajo las propiedades de modelado obtenidas en la propuesta de orden 1 sobre relación de traducción. Los compresores presentados se refieren como $\texttt{TRC}_{(k;1)}$, donde k representa el orden del modelo utilizado para la representación de $\mathcal{L}$ y 1 indica que $\mathcal{R}$ se codifica sobre el modelo de orden 1 construido a partir de la relación de traducción derivada de la secuencia de bipalabras obtenida para la representación del bitexto.

El algoritmo 6.1 muestra una visión genérica del proceso de compresión sobre el que se construyen las técnicas `TRC`. Cómo puede observarse en dicho algoritmo, las técnicas actuales contemplan un proceso semi-estático que implementa dos pasadas completas sobre la secuencia de bipalabras: $\mathcal{B}$.

La primera pasada (descrita entre las líneas 4 y 10) obtiene los vocabularios Σ_L y Σ_R y, a partir de ellos, construye los vocabularios de traducción utilizados en la fase de codificación. La configuración final de cada uno de estos vocabularios se obtiene a partir de las llamadas indicadas entre las líneas $12 - 16$. Por su parte, en las líneas 6 y 7 se gestiona la creación de un nuevo vocabulario para cada nueva palabra identificada en Σ_L. Los métodos $\texttt{actualizar}_{\mathcal{L}}$

y `actualizar`$_{\mathcal{R}}$ se responsabilizan, respectivamente, del modelado de los textos $\mathcal{L}$ y $\mathcal{R}$. En este punto se asume `actualizar`$_{\mathcal{L}}$ como una caja negra dado que depende del método de compresión seleccionado para $\mathcal{L}$ y nos centraremos en el análisis de `actualizar`$_{\mathcal{R}}$ (detallado en el algoritmo 6.2).

El método `actualizar`$_{\mathcal{R}}$ modela de $\mathcal{R}$ atendiendo a su relación de traducción con $\mathcal{L}$. Recibe como parámetro la bipalabra b procesada en cada paso. En primer lugar comprueba que la palabra en $\mathcal{R}$ ($b.meta$) está ya almacenada en Σ_R y, de no estarlo, la introduce como un nuevo elemento. A continuación se centra en la actualización de $b.meta$ en $\tau(b.origen)$, añadiéndola de no estar previamente almacenada. Este proceso de actualización consiste en incrementar el número de ocurrencias de $b.meta$ en $\tau(b.origen)$ de cara a disponer de las estadísticas necesarias para la ordenación final que se utilizará como base para la construcción de la función de diccionario $\mathcal{D}_{b.origen}$ indicada en la sección anterior.

La implementación de Σ_R se lleva a cabo de forma sencilla sobre una tabla hash de pares *(palabra, identificador)*. La propia palabra se utiliza como clave de entrada a la tabla, mientras que el identificador almacenado para cada una de ellas como índice de entrada a un vector (V_R) en el que se almacena información estadística de la palabra. El coste de almacenar Σ_R es $O(\sigma_R) + \sigma_R$.

Por su parte, la implementación de Σ_L conlleva, a su vez, la gestión de los vocabularios de traducción. Al igual que para el caso anterior, se construye una tabla hash de pares *(palabra, identificador)* con las mismas propiedades anteriormente descritas. Sin embargo, el vector actual (V_L) contiene, para cada palabra, una referencia a su vocabulario de traducción. Supongamos que la función `map(p)` devuelve el identificador asociado a la palabra `p` con independencia de que ésta pertenezca a Σ_L o Σ_R. Esto supone que dado $p_L \in \mathcal{L} \wedge$ `map(`p_L`)`=i, la posición $V_L[i]$ contiene un apuntador a un segundo vector (V_τ^i) que almacena el vocabulario $\tau(p_L)$. Cada posición en V_τ^i contiene un valor k / `map(`p_R`)`=k $\wedge$ $(p_L, p_R) \in \mathcal{B}$ y el número de ocurrencias (`occs`) de la bipalabra (p_L, p_R) que representa. El coste de almacenar estas estructuras es $O(\sigma_L) + \sigma_L O(\sigma_R)$. Nótese, finalmente, que los vectores V_τ^i se mantienen ordenados por el valor k, de tal forma que las operaciones de inserción y búsqueda se llevan a cabo en tiempo logarítmico.

Una vez finalizada la primera pasada sobre el bitexto se dispone de la información necesaria para la construcción de las funciones de diccionario asociadas a los vocabularios de traducción. Las configuraciones finales de Σ_L y Σ_R se codifican, independientemente, utilizando un compresor `PPM` aplicado sobre la secuencia de caracteres obtenida al concatenar, ordenadamente, las palabras que forman ambos vocabularios.

A continuación se procesan, uno a uno, el conjunto de vocabularios de traducción. Cada uno de ellos se reordena (*quicksort*) atendiendo al valor `occs` de cada elemento, de tal forma que las traducciones más frecuentes en cada vocabulario pasan a representarse en sus primeras posiciones. Sobre la ordenación obtenida se aplica la heurística presentada en la Sección §4.3.4 de tal forma que se descartan, en cada vocabulario, aquellas traducciones con una única ocurrencia. Esta decisión se toma considerando que aquellas traducciones que se utilizan una sóla vez corresponden con situaciones en las que el alineador textual no ha sido capaz de establecer una traducción "correcta". Cada vocabulario τ se codifica atendiendo a la ordenación final de los identificadores de las palabras de Σ_R que contiene. El proceso de compresión se ejecuta de forma global sobre el conjunto de vocabularios utilizando un código de `Huffman` orientado a bit. El coste temporal del procesamiento de los vocabularios de traducción es $\sigma_L(\Theta(\overline{t}\log\overline{t}) + O(\overline{t}))$, donde $\overline{t}$ es $O(\sigma_R)$ dado que representa el número medio de elementos en τ.

La figura 6.5 muestra visualmente un ejemplo del modelado obtenido para $\mathcal{R}$ sobre las propiedades anteriores. Para simplificar la figura, cada posición en los vectores contiene la palabra que representa aunque ésta se encuentra almacenada en la tabla hash correspondiente. El vocabulario situado en la parte izquierda de la figura representa Σ_R (asociado en este caso al texto

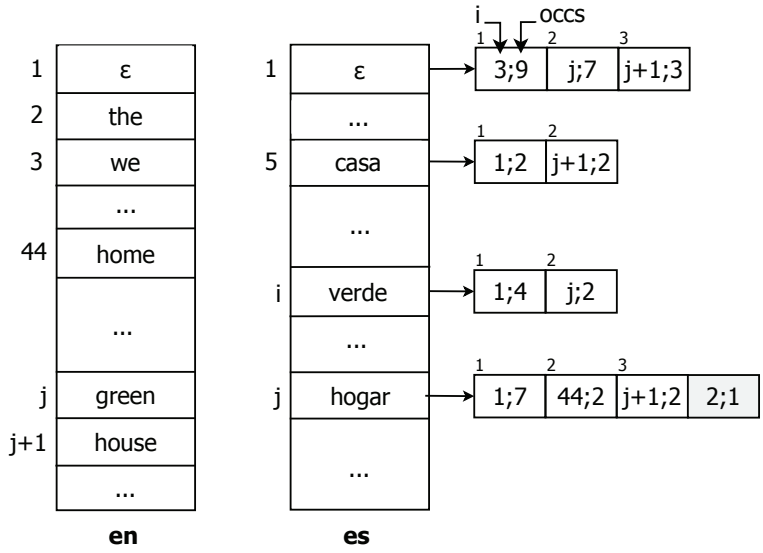


Figura 6.5: Ejemplo de modelado para los compresores $\text{TRC}_{(k;1)}$.

inglés: `en`). Por su parte, el vocabulario de la parte derecha organiza las palabras en Σ_L (en representación del texto en castellano: `es`). Cada entrada en Σ_L apunta a su vocabulario de traducción ya ordenado por el número de ocurrencias de cada una de las traducciones. Las celdas sombreadas representan traducciones con una única aparición y que, por tanto, se descartan de acuerdo a lo planteado previamente.

Si nos fijamos en la palabra "`hogar`", se puede observar como τ("`hogar`")= {"ϵ", "`home`", "`house`", "`the`"}. El último elemento en τ (que representa la bipalabra `(hogar,the)`) se descarta en la función de diccionario de acuerdo a la heurística aplicada. Esto supone que la codificación, en $\mathcal{R}$, de la única ocurrencia de la bipalabra `(hogar,the)` se representa a través del valor `map(the)=k` previamente "escapado" de acuerdo a lo indicado en la Sección §4.3.4. Por lo tanto, la función de diccionario $\mathcal{D}_{hogar}$ finalmente obtenida se define como:

$$\mathcal{D}_{hogar} = \begin{cases} \epsilon \rightarrow 1 \\ \texttt{home} \rightarrow 2 \\ \texttt{house} \rightarrow 3 \end{cases}$$

de tal forma que ϵ se representará como 1 en cada ocurrencia de `(hogar, ϵ)`, "`home`" como 2 (en `(hogar, home)`) y "`house`" como 3 (en `(hogar, house)`).

Se prosigue con el análisis del algoritmo 6.1 en la línea 19 en la que se inicia la segunda pasada sobre el bitexto. El bucle actual se centra en la codificación independiente de $\mathcal{L}$ y $\mathcal{R}$. Al igual que anteriormente, la función `codificar_`$\mathcal{L}$ depende de la técnica de compresión seleccionada para $\mathcal{L}$. Por su parte, la función `codificar_`$\mathcal{R}$ se centra en la compresión de $\mathcal{R}$ utilizando para ello los σ_L vocabularios de traducción obtenidos. De esta manera, cada palabra $p_R \in \mathcal{R}$ se codifica utilizando su identificador en el vocabulario $\tau(p_L)$ establecido por la bipalabra (p_L, p_R) en la que se localiza. Esta secuencia de valores se comprime con un único código de `Huffman`. Nótese como esta distribución es altamente sesgada para los primeros valores en $\mathbb{N}$ dado que las traducciones más frecuentes se ordenan en las primeras posiciones de cada uno de los vocabularios. Esta propiedad es la que permite alcanzar los niveles de efectividad comentados en la Sección §6.5. Finalmente, la codificación de cada bipalabra culmina con la representación de su desplazamiento (de acuerdo a lo detallado en la Sección §6.4.2).

A continuación se plantean dos propuestas para el modelado y codificación de $\mathcal{L}$ en el algoritmo anterior. Esto supone la materialización de todas las operaciones previamente consideradas como cajas negras. La implementación de cada una de estas propuestas, dentro del algoritmo TRC, conlleva la obtención de sendas técnicas de compresión referidas como $\texttt{TRC}_{(0;1)}$ y $\texttt{TRC}_{(1;1)}$.

$\texttt{TRC}_{(0;1)}$ plantea un modelo de orden 0 basado en diccionario para la representación del texto $\mathcal{L}$. Esto supone la necesidad de definir una función de diccionario $\mathcal{D}_L$ equivalente a la planteada en 2V para el mapeo de Σ_L. De esta manera, la función $\mathcal{D}_L$ actual se define como: $\mathcal{D}_L = \{\Sigma_L \rightarrow \mathbb{N}\}$. La técnica obtenida, sobre codificación de Huffman, es equivalente a la presentada en [TM97] adaptada para la gestión de ϵ.

$\texttt{TRC}_{(1;1)}$ adapta la técnica $\texttt{E-G}_1$ (ver §5.3) para la obtención de un modelo de orden 1 para $\mathcal{L}$. La única variación respecto a la técnica original radica en la gestión de ϵ.

La palabra ϵ se utiliza con el fin de transformar el bitexto original en una secuencia de bipalabras. Sin embargo, al analizar $\mathcal{L}$ puede observarse como el uso de ϵ deteriora sus propiedades como lenguaje natural. Supongamos varios alineamientos: (prefiero,i like), (ϵ,would), y (volver,to go back). Si reconstruimos $\mathcal{L}$ a partir de ellos se obtendría "prefiero ϵ volver" cuando lo que realmente se esperaría (desde la perspectiva del lenguaje natural) es "prefiero volver".

Para evitar que el modelo $\texttt{E-G}_1$ se vea perjudicado por este hecho se opta por un gestión alternativa de todas aquellas transiciones en las que interviene la palabra ϵ. El grafo utilizado para modelar $\mathcal{L}$ no considera la representación de un nodo para la palabra ϵ sino que plantea dos decisiones complementarias para la codificación de sus ocurrencias:

(w_1,ϵ): las transiciones en las que ϵ es la palabra *destino* se representan mediante un código de escape EPSILON. Cada nodo en el grafo gestiona este nuevo código de forma similar a como lo hace con NEWVERTEX y NEWEDGE. De esta manera, cada transición del tipo "prefiero ϵ" se codifica a través del valor del código EPSILON en la palabra origen (en este caso 'prefiero'). Es importante destacar que tras esta codificación se mantiene el nodo contexto actual, de tal forma que éste es el utilizado para la codificación de la siguiente palabra, diferente de ϵ, encontrada en $\mathcal{L}$.

(ϵ,w_2): la codificación de las transiciones en las que ϵ actúa como *origen* se lleva a cabo sobre lo comentado en el punto anterior. Esto es, w_2 se representa utilizando como contexto el nodo de la última palabra procesada diferente de ϵ. En el ejemplo actual, la transición "ϵ volver" se codificaría utilizando como contexto el nodo que representa la palabra "prefiero". Esto supone que la transición "prefiero volver" es la realmente representada, lo cual permite mantener la relación original entre las palabras que forman $\mathcal{L}$.

La tabla 6.1 muestra un resumen de los coste temporales de cada una de las operaciones considerada en el algoritmo de compresión TRC. La constante $\overline{t}$ se interpreta como el número medio de elementos en cada vocabulario de traducción. Como puede observarse en la tabla, el coste de las operaciones es similar para $\texttt{TRC}_{(0;1)}$ y $\texttt{TRC}_{(1;1)}$, excepto para codificar_Σ_L.

Finalmente, indicar que el proceso de descompresión es simétrico al actual. De esta manera, en un primer paso se cargan los vocabularios Σ_L y Σ_R junto con los σ_L vocabularios de traducción. Para $\texttt{TRC}_{(1;1)}$ el proceso de carga es ligeramente diferente dado que Σ_L se reconstruye progresivamente al tiempo que se procesa el texto $\mathcal{L}$ sobre el algoritmo de descompresión $\texttt{E-G}_1$ (ver algoritmo 5.2) modificado con la adaptación explicada para la gestión de la palabra ϵ.

Operación	$\texttt{TRC}_{(0;1)}$	$\texttt{TRC}_{(1;1)}$
`actualizar_`$\mathcal{L}$	$O(1)$	
`actualizar_`$\mathcal{R}$	$O(\log \overline{t})$	
`procesar_`τ	$\Theta(\overline{t} \log \overline{t}) + O(\overline{t})$	
`codificar_`$\mathcal{L}$	$O(1)$	$O(\log \sigma_L)$
`codificar_`$\mathcal{R}$	$O(1)$	
`codificar_desplazamiento`	$O(1)$	

Tabla 6.1: Análisis de costes de las operaciones requeridas en el algoritmo de compresión `TRC`.

6.4.4. 2-Level Compressor for Aligned Bitexts (2LCAB)

`2LCAB` [ABMPSM09] plantea una técnica de compresión de bitextos basada en su representación sobre un modelo de bipalabras. Esto significa, como se explica en la sección anterior, que cada bipalabra en la secuencia se modela y codifica mediante un único símbolo de tal forma que `2LCAB` es capaz de representar el significado compartido entre las dos palabras que la constituyen y, por tanto, asumir como unidad simbólica el *concepto* derivado de la relación de traducción. Esta nueva perspectiva ofrece una representación más realista de la naturaleza del bitexto a costa de añadir una serie de propiedades semánticas (polisemia y sinonimia) que deben ser adecuadamente gestionadas con el objetivo de no deteriorar la efectividad de la técnica de compresión.

La bipalabra tiene una naturaleza semántica como símbolo de representación pero, a su vez, conserva propiedades estructurales derivadas de las dos palabras que la forman. Esto supone que el concepto de bipalabra puede ser analizado desde dos puntos de vista complementarios:

- Desde una perspectiva *estructural*, la bipalabra está formada por la combinación de dos palabras pertenecientes a idiomas diferentes. Esto supone, como se planteaba anteriormente, que cada una de las palabras puede considerarse como un símbolo independiente dentro del subconjunto determinado por las propiedades del texto en el que se identifica.

- Desde una perspectiva *semántica*, la bipalabra simboliza un determinado significado dentro del ámbito que representa un bitexto.

El diseño de `2LCAB` considera la experiencia obtenida con los modelos de representación basados en palabras. Para el caso actual se elige una propuesta equivalente a `2V` teniendo en cuenta que el objetivo de `2LCAB` no sólo radica en la compresión del bitexto sino que además enfoca el problema de la manipulación directa de sus contenidos sobre la representación comprimida. Este hecho justifica la necesidad de representar, independientemente, Σ_L y Σ_R y a partir de ellos obtener, en el nivel superior, una representación de Σ_B que permita utilizar la información semántica contenida en las bipalabras en el diseño de operaciones sobre el bitexto comprimido.

La figura 6.6 muestra una descripción conceptual del esquema seguido por `2LCAB`. En primer lugar cabe destacar la diferencia existente entre el enfoque en dos niveles planteado por `2LCAB` y el de las técnicas `TRC`. En el caso actual, el segundo nivel plantea una única función de diccionario sobre Σ_B frente a las σ_L funciones manejadas en `TRC` para cada τ.

`2LCAB` almacena, en el primer nivel, las representaciones de Σ_L y Σ_R. El proceso de identificación de las palabras es similar al desarrollado en `TRC` mientras que el criterio seguido para la ordenación de estos vocabularios considera que los códigos asignados a las palabras son, posteriormente, utilizados en la representación de las bipalabras en Σ_R. De esta manera, cada uno de los vocabularios se reordena atendiendo al número de bipalabras diferentes en las que se utiliza cada palabra. Esta decisión permite representar de forma más efectiva tanto las palabras polisémicas como aquellas que tienen muchos sinónimos y que, por tanto, aparecen en un mayor

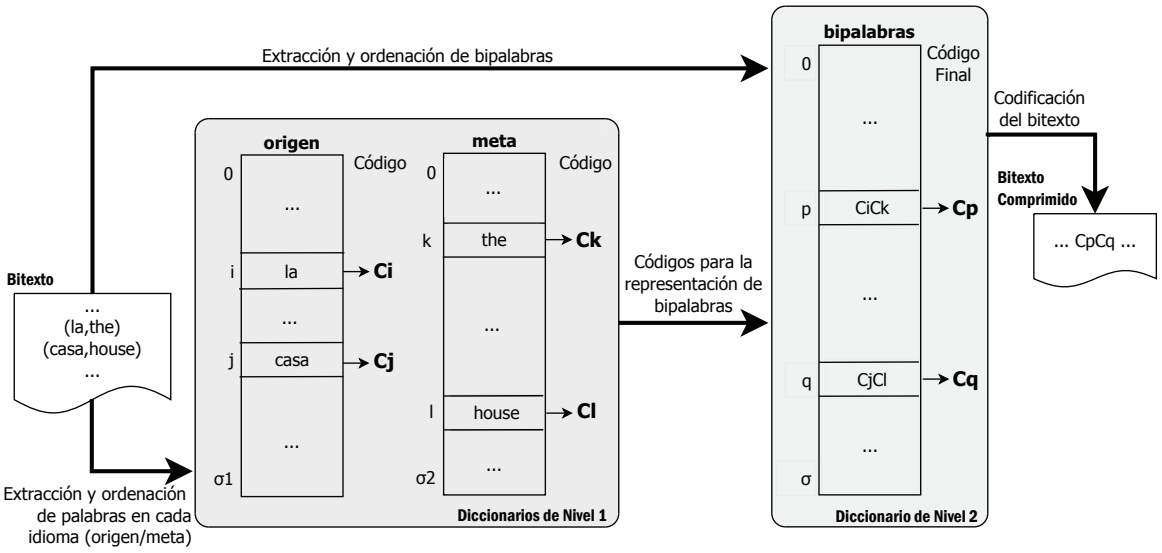


Figura 6.6: Descripción conceptual del compresor 2LCAB.

número de bipalabras. Para la codificación de cada palabra se utiliza el código ETDC que identifica su posición en el diccionario correspondiente. Este código se obtiene mediante la función `encode(i)` asociada con dicho esquema de codificación (ver §3.3.3).

El vocabulario de segundo nivel plantea una representación compacta de Σ_B. Esta representación se construye a partir de la concatenación de los códigos utilizados, en el primer nivel, para la identificación de las palabras que constituyen cada una de las bipalabras. Supongamos la bipalabra `(casa,house)` mostrada en la figura anterior. Sus palabras componentes, `casa` y `house` se representan, respectivamente, como $\texttt{C}_j$`=encode(j)` y $\texttt{C}_l$`=encode(l)`. Esto supone que la presente bipalabra, en el vocabulario de segundo nivel, se representa por el código C_jC_l obtenido por la concatenación de los dos códigos anteriores. La ordenación de Σ_B sigue un criterio de frecuencia de tal forma que las bipalabras más frecuentes en el bitexto se sitúan en las primeras posiciones. La codificación de las bipalabras se lleva a cabo sobre los principios indicados para los vocabularios de primer nivel. Esto supone utilizar el código ETDC obtenido de acuerdo a la posición ocupada en Σ_B por cada bipalabra.

El proceso de compresión sustituye cada bipalabra en el bitexto por el código ETDC obtenido sobre la configuración final de Σ_B. El resultado final de la compresión se distribuye sobre cuatro flujos independientes: $\texttt{Voc}_L$ y $\texttt{Voc}_R$ contienen las cadenas de texto que representan las palabras ordenadas en cada vocabulario, $\texttt{Voc}_B$ almacena la representación compacta de las bipalabras ordenadas de acuerdo a su frecuencia de aparición y, finalmente, el propio bitexto comprimido de acuerdo al proceso de sustitución comentado. Esto supone que tanto $\texttt{Voc}_L$ como $\texttt{Voc}_R$ son flujos textuales mientras que $\texttt{Voc}_B$ y el bitexto comprimido son secuencias de códigos ETDC.

El algoritmo 6.3 muestra una descripción del proceso de compresión desarrollado por 2LCAB. Su estructura y propiedades son muy similares a las planteadas en TRC. La primera pasada sobre el bitexto (representada entre las líneas $5-12$) obtienen los vocabularios Σ_L, Σ_R y Σ_B almacenando la información estadística necesaria para su posterior ordenación. El único método no comentado en TRC es `actualizar_`$\mathcal{B}$ centrado en la gestión de Σ_B.

Los métodos de procesamiento de los vocabularios (líneas $14-16$) reordenan y codifican cada uno de los vocabularios, obteniendo una función de diccionario independiente para cada uno de ellos. Finalmente, la segunda pasada representa cada bipalabra de acuerdo a la función de diccionario obtenida en Σ_B, reemplazando dicha bipalabra por su código identificador en Σ_B. Como paso final, en el procesamiento de cada bipalabra, se codifica su valor de desplazamiento

Algoritmo 6.3 Compresión 2LCAB

```
1: Σ_L ← ∅;
2: Σ_R ← ∅;
3: Σ_B ← ∅;
4:
5: for all b ∈ B do
6:    if b.origen ∉ Σ_L then
7:       τ(b.origen) ← ∅;
8:    end if
9:    actualizar_L(b.origen);
10:   actualizar_R(b.meta);
11:   actualizar_B(b);
12: end for
13:
14: procesar_Σ_L();
15: procesar_Σ_R();
16: procesar_Σ_B();
17:
18: for all b ∈ B do
19:   codificar_B(b);
20:   codificar_desplazamiento(b.desp);
21: end for
```

de acuerdo al esquema presentado a este respecto (ver §6.4.2).

Al igual que `TRC`, el proceso de descompresión es simétrico al actual. En primer lugar se reconstruyen los diccionarios de palabras y bipalabras. La lectura secuencial del bitexto comprimido permite obtener, en cada paso, el código `ETDC` que indica la posición en el vocabulario de la siguiente bipalabra que debe ser añadida a la salida. Dicha posición j es obtenida como `j=decode(C`$_j$`)`.

En lo que respecta al coste de las operaciones, éste es muy similar al presentado para `TRC`. Únicamente la operación `actualizar_`$\mathcal{R}$ pasa a tener un coste $O(1)$ dado que se puede acceder a la palabra mediante la tabla hash considerada para Σ_R de palabras. Por su parte, las nuevas operaciones centradas en la gestión y procesamiento de Σ_B presentan un coste comparable a las operaciones equivalente en los otros vocabularios. Esto es, `actualizar_`$\mathcal{B}$ y `procesar_`Σ_B tienen costes, respectivamente, $O(1)$ y $\Theta(\sigma_B \log \sigma_B)$. Finalmente, `codificar_`$\mathcal{B}$ tiene un coste $O(1)$.

`2LCAB` utiliza, como se ha comentado, un esquema de codificación orientado a byte sobre la técnica `ETDC`. Esta propiedad se utiliza como base para el desarrollo de las operaciones consideradas a continuación. Sin embargo, en la sección siguiente, se plantea también una implementación de `2LCAB` basada en un código canónico de `Huffman` con el fin de poder comparar, en igualdad de condiciones, la efectividad de la técnica actual respecto a los compresores `TRC`.

Operaciones de búsqueda sobre el bitexto comprimido. El resultado de la compresión `2LCAB` está formado tanto por la representación de Σ_L, Σ_R y Σ_B, como por el propio bitexto comprimido sobre codificación `ETDC`. Las propiedades de este resultado permiten un acceso aleatorio al bitexto comprimido así como la utilización de algoritmos de *pattern-matching* tipo Boyer-Moore [BM77] para la localización de patrones de búsqueda. El siguiente conjunto de operaciones pueden llevarse a cabo, actualmente, sobre un alineamiento $1:1$`No Desp.`

El primer bit de cada byte utilizado en la codificación `ETDC` permite distinguir si éste es un

stopper (byte final de una palabra de código) o un *continuer* (byte no final de una palabra de código). Esta propiedad es suficiente para la ejecución de los algoritmos de *pattern-matching* referidos dado que facilita la comprobación de que una determinada ocurrencia se corresponde con el código buscado o, por el contrario, es sólo un sufijo de éste. Esta sencilla comprobación consiste en evaluar el valor del byte previo al identificado como posible resultado. Si este byte se corresponde con un *continuer* entonces el resultado localizado constituye un falso positivo mientras que si el byte previo es un *stopper* se tiene la seguridad de haber localizado un resultado válido. Esta sobrecarga de cómputo es despreciable dentro del proceso global considerando que esta comprobación sólo es necesaria en el caso de localizar un posible resultado lo cual es infrecuente [BFNP07].

Algunas de las aplicaciones en las que se considera el uso de los bitextos no requieren su completa descompresión sino que únicamente necesitan la extracción de pequeños fragmentos del mismo (*snippets*). Las propiedades actuales de `2LCAB` facilitan la ejecución de un conjunto de estas operaciones sobre el bitexto comprimido:

- *Búsqueda de traducciones*: identifica, para una palabra dada, cuáles son las palabras por las que se traduce en el texto complementario. Dadas $p_L \in \Sigma_L$ y $p_R \in \Sigma_R$, la función actual $\texttt{Busq}_{trads}$ se define como:

$$Busq_{trads}(p_L) = \{p_x \in \Sigma_R \ / \ (p_L, p_x) \in \Sigma_B\}$$

$$Busq_{trads}(p_R) = \{p_y \in \Sigma_L \ / \ (p_y, p_R) \in \Sigma_B\}$$

- *Búsqueda de palabras*: permite localizar todas las ocurrencias de una determinada palabra en su idioma correspondiente. Dadas $p_L \in \Sigma_L$ y $p_R \in \Sigma_R$, la función actual $\texttt{Busq}_{pals}$ se define como:

$$Busq_{pals}(p_L) = \{\mathcal{B}[i] \ \forall i \in [1,n] \ / \ \mathcal{B}[i] = (p_L, p_x) \in \Sigma_B\}$$

$$Busq_{pals}(p_R) = \{\mathcal{B}[j] \ \forall j \in [1,n] \ / \ \mathcal{B}[j] = (p_y, p_R) \in \Sigma_B\}$$

- *Búsqueda de bipalabras* (búsqueda conceptual): permite localizar todas las ocurrencias de una determinada bipalabra en el bitexto. Dadas $p_L \in \Sigma_L$ y $p_R \in \Sigma_R$, tal que $p_B = (p_L, p_R) \in \Sigma_B$, la función actual $\texttt{Busq}_{bipals}$ se define como:

$$Busq_{bipals}(p_B) = \{\mathcal{B}[k] \ \forall k \in [1,n] \ / \ \mathcal{B}[k] = p_B \in \Sigma_B\}$$

- *Extracción de snippets*: facilita recuperar un fragmento del bitexto (de una determinada longitud) en cualquiera de los idiomas que lo forman.

Para el procesamiento del texto comprimido se requiere un almacenamiento y gestión eficiente de los vocabularios obtenidos en la compresión del bitexto. Para los vocabularios de primer nivel se procesan los flujos $\texttt{Voc}_L$ y $\texttt{Voc}_R$ y se construyen sendas tablas hash en las que se almacena, para cada palabra, su identificador correspondiente. Este valor identificativo i se utiliza tanto para obtener el código con el que la palabra se representa en Σ_B como para acceder a la propia palabra almacenada en la i-ésima posición del vector correspondiente al vocabulario de su idioma. La obtención del identificador asociado a una palabra se ejecuta en tiempo $O(1)$.

El vocabulario de bipalabras Σ_B se obtiene a partir del flujo $\texttt{Voc}_B$ y se almacena en un vector que contiene, en cada posición, los códigos identificadores de las palabras que componen la bipalabra. Durante este procesamiento se crea un *bitmap* auxiliar que facilita la ejecución de las operaciones sobre Σ_B: los valores '0' en el *bitmap* se corresponden con aquellas posiciones de Σ_B que contienen bytes *continuer* mientras que los bits '1' se corresponde con las posiciones ocupadas por los bytes *stopper*. Considerando las σ_B bipalabras diferentes que forman Σ_B, el coste del presente *bitmap* es $O(\overline{l}b)$, donde $\overline{l}$ es la longitud media, en bytes, de los códigos utilizados para

Algoritmo 6.4 Búsqueda de traducciones

Require: $palabra, idioma$;
1: $traducciones \leftarrow \emptyset$;
2:
3: **if** $idioma = 1$ **then**
4: $id \leftarrow \Sigma_L.buscar(palabra)$;
5: **else**
6: $id \leftarrow \Sigma_R.buscar(palabra)$;
7: **end if**
8:
9: $C \leftarrow encode(id)$;
10:
11: **for all** $C \in \Sigma_B$ **do**
12: $pos \leftarrow bitmap.rank_1(pos)$;
13: **if** $pos \ mod \ 2 = idioma$ **then**
14: $traducciones \leftarrow traducciones \cup \{pos/2\}$;
15: **end if**
16: **end for**

la compactación de las bipalabras. En términos prácticos, las palabras se identifican en el primer nivel con códigos de 1 a 3 bytes, lo que supone que, en el segundo nivel, se utilicen secuencias de 2 a 6 bytes para la representación de cada una de las bipalabras. La implementación de este *bitmap* se desarrolla sobre la propuesta de González, *et al.* [GGMN05] que requiere un 5 % de espacio adicional sobre el tamaño original del *bitmap* y resuelve consultas *rank*/*select* en tiempo constante.

Búsqueda de traducciones. La operación actual permite obtener todas las traducciones para una palabra dada sin necesidad de acceder al bitexto comprimido. Toda la información necesaria para la resolución de esta consulta se encuentra disponible en la estructura de vocabularios.

El algoritmo 6.4 describe los pasos seguidos para completar la búsqueda de traducciones. El método actual recibe como parámetros la propia palabra y el idioma al que ésta corresponde (en el presente algoritmo se asume que `1` representa el idioma $\mathcal{L}$ y `0` al $\mathcal{R}$). En primer lugar se recupera el identificador de la palabra a través de una consulta a la tabla hash que almacena el vocabulario del idioma indicado. A partir de este valor se obtiene su código `ETDC` (referido como C) que se suministra, como entrada, a un algoritmo de *pattern-matching* exacto que se ejecuta sobre la representación comprimida de Σ_B. El bucle *for* representa la ejecución del algoritmo de *pattern-matching* que, en cada iteración, devuelve la posición *pos* en la que identifica un posible resultado. Este valor *pos* requiere una comprobación, sobre el bitmap auxiliar, considerando que el código C puede aparecer como representación de una palabra de cualquiera de los idiomas. Supongamos un posible resultado en la posición *pos* de Σ_B. La operación `rank`$_1$`(pos)` cuenta el número de bits '1' existentes en el *bitmap* hasta la posición *pos*; este valor identifica el número de palabras representadas hasta esta posición. De esta manera, un valor impar indica que el código actual representa una palabra perteneciente a Σ_L mientras que un valor par refiere una palabra en Σ_R. Si el posible resultado pertenece al idioma indicado se tiene la seguridad de haber encontrado la palabra solicitada. El siguiente paso es identificar la bipalabra que la contiene. Este cálculo se obtiene de forma sencilla contando el número de bipalabras representadas hasta *pos*: `rank`$_1$`(pos)/2`. El resultado se añade al conjunto de traducciones previamente encontradas.

La ejecución del presente método obtiene un resultado final formado por el identificador

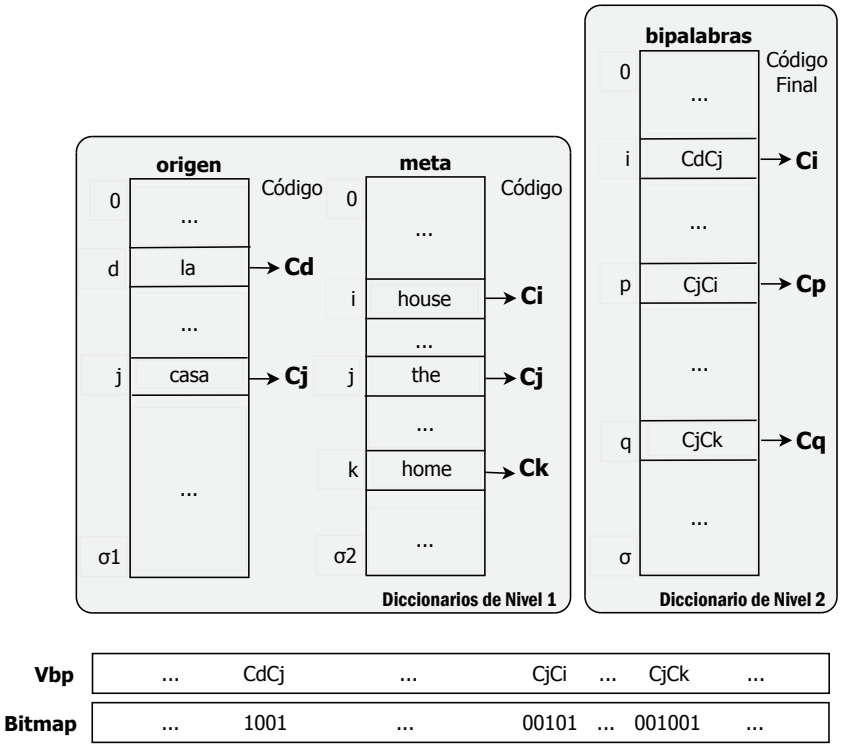


Figura 6.7: Ejemplo para la búsqueda de traducciones.

numérico de cada una de las bipalabras en las que se utiliza la palabra solicitada. El conjunto de traducciones puede ser fácilmente obtenido en Σ_B accediendo a las posiciones indicadas por el conjunto de identificadores y recuperando, en cada posición, la palabra complementaria a la solicitada. El coste de la esta operación es $O(\sigma_b)$.

La figura 6.7 muestra un ejemplo de configuración de bitexto comprimido con `2LCAB` que se utiliza para facilitar el entendimiento del proceso anterior. Se asume que el código C_d tiene longitud 1 byte, C_i 2 bytes y C_j,C_k 3 bytes. Se omite la representación de las tablas hash utilizadas en la implementación de Σ_L y Σ_R.

Supongamos que se ejecuta la operación `buscar_traducciones`(“`casa`”,`1`) para recuperar todas las traducciones de la palabra `casa`$\in \Sigma_L$. En primer lugar se consulta la tabla hash de Σ_L y se recupera el valor j que identifica la palabra “`casa`”. Aplicando la función `encode` se obtiene el código $\mathtt{C}_j$ utilizado en la representación de las bipalabras en las que se utiliza la palabra solicitada. El algoritmo de *pattern-matching* localiza 3 ocurrencias del patrón C_j en Σ_B:

- La primera de ellas se localiza en la posición pos_1. Al ejecutar la operación `rank_1`(pos_1) se obtendría el valor $2i$, por tanto un valor par. Esto significa que el código localizado identifica a una palabra perteneciente a Σ_R y por tanto es un falso positivo.

- La segunda ocurrencia se localiza en la posición pos_2. La operación `rank_1`(pos_2) devuelve el valor impar $2p-1$. Esto indica que la palabra pertenece a Σ_L y que está dentro de la bipalabra identificada como p.

- La tercera ocurrencia, localizada en la posición pos_3, se explica igual que la anterior e identifica la bipalabra q.

El resultado de la consulta actual son los valores p, q. Esto supone que las traducciones de “`casa`” están codificadas en la palabra meta de las bipalabras almacenadas en las posiciones p

y q de Σ_B. Se recuperan los códigos C_i y C_k que son decodificados en i y k de tal forma que las traducciones de "`casa`" son "`house`" y "`home`", respectivamente almacenadas en las posiciones i y k de Σ_R.

Búsqueda de palabras. La operación actual localiza todas las ocurrencias, en el bitexto, de la palabra solicitada con independencia del idioma al que ésta pertenezca. Nótese que la representación comprimida del bitexto sobre `2LCAB` es orientada a bipalabras por lo que la búsqueda de una palabra requiere localizar todas las ocurrencias de las diferentes bipalabras en las que ésta se utiliza. Esto supone que la operación actual está estructurada en dos partes claramente diferenciadas.

En primer lugar se necesita identificar todas las bipalabras en las que se utiliza la palabra solicitada. Este conjunto de identificadores se obtiene mediante la operación anterior `buscar_traduccion`. Cada uno de estos identificadores se transforma en su código `ETDC` mediante la operación `encode()` y se añade a un *trie* de búsqueda que se utilizado como entrada a un algoritmo de *pattern-matching* múltiple. El algoritmo `SetHorspool` [Hor80, NR02] es el elegido para el caso actual considerando que la representación comprimida del bitexto utiliza códigos de, a lo sumo, 3-4 bytes cuya distribución de frecuencia es relativamente uniforme en $[0, 255]$. La ejecución de `SetHorspool`, sobre el *trie* de bipalabras, obtiene todas las posiciones del bitexto en el que estas bipalabras se utilizan, lo que supone identificar todas las ocurrencias de la palabra solicitada en cada una de sus posibles traducciones.

La primera parte del proceso actual se lleva a cabo en un tiempo $O(\sigma_B)$ mientras que la ejecución del algoritmo `SetHorspool` tiene un coste $O(n)$, considerando un total de n bipalabras representadas en el bitexto comprimido.

Búsqueda de bipalabras. La búsqueda de bipalabras representa una simplificación de la operación anterior. Para el caso actual, el *trie* de entrada al algoritmo `SetHorspool` está formado por un único código: el que representa la bipalabra buscada. El interés propio de este tipo búsqueda radica en su significado.

La búsqueda de una bipalabra (p_L,p_R) localiza, en primer lugar, los identificadores de $\mathtt{p}_L$ y $\mathtt{p}_R$ en Σ_L y Σ_R. Mediante la función `encode` se obtienen sus códigos C_L y C_R y se concatenan formando el código $C_L C_R$ que representa la bipalabra en Σ_B. Se obtiene el identificador de la bipalabra $C_L C_R$ y se añade al trie sobre el que se ejecuta el algoritmo `SetHorspool` de manera idéntica a la explicada para la búsqueda de palabras anterior.

La operación actual tiene una importancia significativa desde una perspectiva conceptual. La búsqueda de una bipalabra representa la búsqueda etiquetada de una palabra respecto a su traducción en el idioma complementario. Esto implica que el resultado de la operación contiene las ocurrencias de una palabra con un determinado significado, lo cual supone la búsqueda de uno de los conceptos que representa la palabra.

Supongamos la palabra `caña`, previamente comentada como ejemplo de palabra polisémica. Una búsqueda por palabras nos devolvería todas las ocurrencias de la palabra en el bitexto con independencia de su significado. Sin embargo, puede que el interés de la búsqueda esté centrado únicamente en aquellos casos en los que esta palabra se utilice en su acepción como *instrumento de pesca*. La búsqueda de bipalabras permitiría recuperar estas ocurrencia de forma sencilla sin más que solicitar el patrón `(caña,rod)`. Esto supone, desde una perspectiva clásica, que la búsqueda por bipalabras es capaz de desambigüar el significado de la palabra solicitada utilizando para ello la información etiquetada a través de su traducción en el ámbito de la bipalabra.

Extracción de *snippets*. La extracción de *snippets* se utiliza como herramienta para la contextualización de los resultados obtenidos en los procesos de búsqueda dado que es capaz de

recuperar un conjunto de símbolos alrededor de cada ocurrencia localizada. La longitud de los *snippets* puede ajustarse a las necesidades de información existentes en cada contexto.

La implementación de esta operación se lleva a cabo de forma sencilla sobre las propiedades de una codificación `ETDC`, considerando que ésta permite la descompresión del bitexto a partir de cualquier posición del mismo. El proceso de extracción fija la posición en la que éste se inicia y lee, una a una, las k palabras de código que conforman el *snippet*. Por medio de la función `decode` transforma los códigos en los identificadores numéricos que permiten acceder a Σ_B en aquellas posiciones en las que se almacenan los símbolos que conforman el *snippet*. Nótese como con la información disponible se pueden reconstruir los fragmentos de texto asociados a cada uno de los idiomas en el bitexto. Esta es una propiedad importante dado que abstrae el idioma utilizado en la búsqueda del considerado para la extracción del *snippet*. Esto permite que, para una determinada búsqueda, el *snippet* puede ser extraído en el mismo idioma en el que se formula la consulta, en el complementario o en los dos idiomas del bitexto.

A continuación se comenta un ejemplo real de búsqueda en un bitexto `(es-en)`. Se utiliza la palabra `casa` como patrón en el idioma origen y de entre todas sus ocurrencias elegimos las dos que se localizan, respectivamente, en las posiciones $p1$ y $p2$. Los *snippets* extraídos son los siguientes:

$p1$:

```
(es) el erika se hundió delante de mi casa y allí sigue con 20.000
 (en) the erika sank near my home and is still there with 20 000
```

$p2$:

```
(es) momento de que europa ponga su propia casa en orden, de imbuir a
    (en) time for europe to put is own house in order; to instil
```

La ocurrencia localizada en la posición $p1$ se identifica a través de la bipalabra `(casa,home)` mientras que la localizada en $p2$ se identifica mediante la bipalabra `(casa,house)`. Si la búsqueda se centra en obtener las ocurrencias de "`casa`" en su propio idioma los *snippets* obtenidos serían los etiquetados como `(es)`. Si, por el contrario, se solicita encontrar las ocurrencias de las traducciones de "`casa`" en el texto inglés, los *snippets* obtenidos serían los etiquetados como `(en)`. Finalmente, dada la palabra buscada se podrían obtener los contextos originales en los que ésta se utiliza junto a aquellos que contienen sus traducciones. Este tipo de resultados son muy útiles en aplicaciones de apoyo a la traducción dado que facilitan la comparación entre el uso de la palabra en su idioma original y como ésta se traduce en el idioma complementario.

6.5 Experimentación

La presente sección de experimentación afronta los diferentes aspectos tratados a lo largo del capítulo. En primer lugar se lleva a cabo un exhaustivo estudio estadístico sobre las diferentes propiedades que caracterizan la representación de los bitextos utilizados en la experimentación actual. Dicho estudio trata, de forma independiente, el análisis de las relaciones existentes entre diferentes pares de idiomas y el impacto que éstas tienen sobre los modelos de representación (orientados a palabras y bipalabras) considerando, a su vez, las diferentes políticas de alineamiento propuestas.

A continuación se lleva a cabo un estudio de la eficiencia y la efectividad de las diferentes técnicas propuestas. En todo este estudio se utiliza un corpus heterogéneo compuesto por diferentes pares de idiomas. Finalmente, se analiza la eficiencia de las operaciones de búsqueda consideradas para `2LCAB`. El entorno de experimentación en el que se lleva a cabo el presente estudio está completamente detallado en el Apéndice A.

6.5.1. Estudio estadístico

La propiedad fundamental que distingue un bitexto de un texto genérico en lenguaje natural radica en su vocabulario, dado que el bitexto contiene palabras procedentes de dos idiomas diferentes. Como se ha planteado en las secciones anteriores, las propiedades estadísticas del vocabulario, utilizado en el proceso de compresión, varían dependiendo del tipo de símbolo considerado en el modelado del bitexto: palabras o bipalabras. En este segundo caso, el tamaño del vocabulario Σ_B, utilizado en la compresión del bitexto, está influenciado por los vocabularios de palabras Σ_L y Σ_R dado que cada bipalabra combina dos palabras cuyo significado representa una traducción mutua en el bitexto.

En primer lugar se analizan las propiedades de los vocabularios para diferentes pares de idiomas. El estudio actual considera hasta ocho pares de idiomas: alemán-inglés (`de-en`), francés-inglés (`fr-en`), inglés-castellano (`en-es`), castellano-francés (`es-fr`), castellano-italiano (`es-it`), castellano-portugués (`es-pt`), castellano-catalán (`es-ca`) y castellano-gallego (`es-gl`). El conjunto de pares seleccionados nos permite disponer de un entorno heterogéneo de experimentación en el que se consideran tanto idiomas de raíz próxima (castellano, catalán y gallego), otros relativamente cercanos entre sí (castellano respecto a francés, italiano y portugués) y, finalmente, idiomas con propiedades notablemente diferentes (alemán, castellano, inglés y francés). Las estadísticas siguientes se obtienen con bitextos de tamaño, aproximado, de 100 MB. excepto para el par (`es-gl`) cuyos resultados se obtienen a partir de un bitexto de tamaño 10 MB. que es el mayor del que se ha podido disponer.

Aunque el trabajo actual se centra en la representación compacta de bitextos, no debe perderse de vista que el objetivo último de este tipo de contenidos es su utilización en un conjunto de aplicaciones que hacen uso de la información de traducción mutua que éstos contienen. Ante esta situación consideramos estimar la *calidad* del alineamiento o de la traducción a través del porcentaje de utilización de la palabra ϵ. La existencia de altos porcentajes en el uso de ϵ implica un mayor número de palabras que no han podido ser traducidas en su idioma complementario.

Asimismo, la utilización de ϵ tiene un efecto secundario en lo que respecta al número medio de traducciones en los que interviene cada palabra. La traducción de un palabra p por ϵ implica renunciar a su verdadera traducción en el idioma complementario. Si $p \in \Sigma_L$, la relación (p,ϵ) se utiliza, de forma genérica, para la representación de todas las relaciones de traducción que no han podido ser identificadas para la palabra p. La reducción en el uso de ϵ es síntoma de que un mayor número de relaciones de traducción han podido ser establecidas para p, lo cual tiende a aumentar el número de traducciones diferentes asociados con dicha palabra. Desde la perspectiva de `TRC`, mantener un número bajo de traducciones por palabra permite utilizar un menor número de bits para su representación en el contexto planteado por el texto origen. Igualmente, `2LCAB` se beneficia de la existencia de un menor número de pares de palabras representadas como traducción mutua en el bitexto. Por lo tanto, la reducción en la utilización de ϵ y el consiguiente aumento del número medio de traducciones en los que interviene una palabra suponen una penalización para la efectividad esperada en nuestros modelos de representación. La sección actual trata este efecto a través de las resultados obtenidos para los estadísticos anteriores analizando su influencia en las técnicas de compresión.

La experimentación actual enfoca la cuatro políticas de alineamiento previamente comentadas. Por un lado, los alineamientos `1:1` relacionan una palabra en $\mathcal{L}$ con, a lo sumo, una palabra en $\mathcal{R}$ y viceversa. Aquellas palabras, en cualquiera de los dos textos, para las que no se haya identificado su relación de traducción aparecen alineadas con la palabra ϵ. La política `1:1 No Desp.` representa la opción más sencilla de todas, dado que considera sólo aquellos alineamientos *uno a uno* que no produzcan cruces con otros alineamientos. Para afrontar esta última debilidad, se plantea la política `1:1 Desp.` que, aunque mantiene el enfoque de alineamiento *uno a uno*, utiliza un valor de desplazamiento con el fin de minimizar el número de alineamientos descarta-

dos por su cruce sobre otros. Esto supone la posibilidad de identificar relaciones de traducción entre pares de idiomas cuya estructura siga un orden diferente y que, por consiguiente, produzca numerosos alineamientos cruzados. Sin embargo, es un hecho que las relaciones de traducción entre dos textos (con independencia de las propiedades particulares de sus idiomas) no pueden reducirse a alineamientos sencillos *uno a uno*, dado que la semántica de una palabra en $\mathcal{L}$ puede estar repartida entre varias palabras en $\mathcal{R}$ y viceversa. Este tipo de situaciones son las que se cubren a través de alineamientos `1:N` que permiten la utilización de multipalabras en $\mathcal{R}$ como símbolos primitivos de su diccionario. Sobre este tipo de alineamiento se definen dos políticas: `1:N Cont.` limita la formación de multipalabras a grupos de palabras *contiguas* en $\mathcal{R}$. Esta política descarta el alineamiento menos utilizado hasta la multipalabra formada está compuesta exclusivamente por palabras contiguas. Por su parte, `1:N No Cont.` permite que estos grupos de palabras puedan ser identificados como traducciones de una palabra en $\mathcal{L}$ sin la necesidad de que sus palabras componentes sean contiguas en $\mathcal{R}$.

Aunque, como se decía anteriormente, el proceso de experimentación se lleva a cabo sobre ocho pares de idiomas diferentes sólo algunos de ellos se detallan en profundidad de acuerdo a las similitudes identificadas entre diferentes pares de idiomas. Esto nos permite clasificar los bitextos en tres categorías:

1. La primera categoría se centra en pares de idiomas con grandes diferencias gramaticales. Para este caso se considera el bitexto `de-en` sobre el que puede observar como la traducción obtenida mediante alineamientos `1:1` posee una baja calidad debido a la dificultad de relacionar palabras en alemán con palabras únicas en inglés. Sin embargo, la consideración de multipalabras mejora notablemente tanto la calidad de la traducción como las tasas de compresión obtenidas con cada una de las técnicas planteadas.

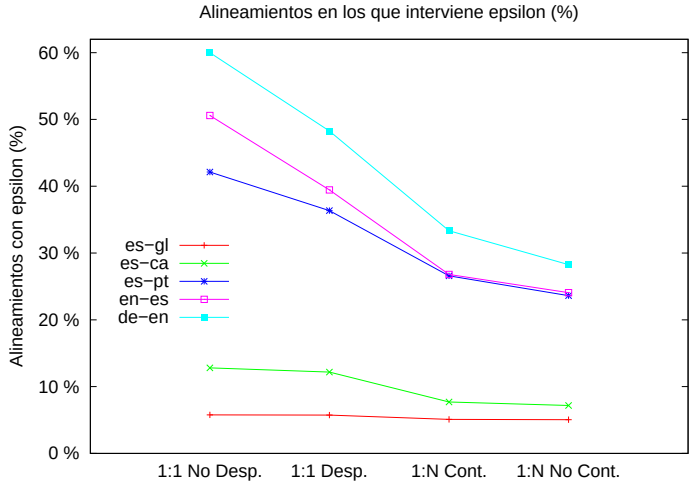
2. Los bitextos `en-es` y `es-pt` se eligen como representantes de aquellos pares de idiomas que, aún manteniendo una raíz común, han desarrollado una evolución distante. Las conclusiones obtenidas para esta categoría se pueden extender a otros pares de idiomas como `fr-en`, `es-fr` o `es-it` que muestran un comportamiento estadístico comparable con la diferencia inherentes a las propiedades específicas de cada par de idiomas.

3. Finalmente, la tecerca categoría estudia la relación existente entre pares de idiomas con una raíz común y una evolución más próxima que la del grupo anterior. En esta categoría se incluyen los idiomas oficiales españoles con raíz latina: *castellano*, *catalán* y *gallego*. Las conclusiones obtenidas para los bitextos en los que intervienen estos idiomas son similares aunque de nuevo caracterizadas por las propiedades específicas de cada par de idiomas. En el estudio que se muestra a continuación puede observarse una mayor proximidad semántica y estructural entre *castellano* y *gallego* que entre *castellano* y *catalán*, aunque en ambos casos sus propiedades estadísticas son notablemente diferentes a las del caso anterior.

Evolución del tamaño de los vocabularios. El tamaño del vocabulario de un bitexto depende del tipo de modelo seleccionado para su representación. De acuerdo a lo planteado en la Sección §6.3, consideramos modelos orientados a palabras y bipalabras. En el apartado actual se plantea un estudio estadístico relativo al tamaño tanto de los vocabularios de palabras como de los de bipalabras. Nótese que el tamaño σ_L del vocabulario Σ_L se mantiene constante para todas las políticas de alineamiento consideradas dado que todas ellas fijan la utilización de un alfabeto orientado a palabras. Por su parte, el tamaño σ_R de Σ_R es similar en los alineamientos *uno a uno* (dado que sólo considera la representación de palabras) y muestra un crecimiento progresivo para los casos *uno a muchos* en los que se extiende el alfabeto de palabras original con multipalabras contiguas y no contiguas.

Bitexto	Palabras				Bipalabras			
	Σ_L	Σ_R			Σ_B			
	1:1	1:1	1:N		1:1		1:N	
			Cont.	No Cont.	No Desp.	Desp.	Cont.	No Cont.
`de-en`	139010	51015	201511	460100	387985	390210	660466	789381
`en-es`	51352	81868	177244	312667	367857	366387	559142	618586
`es-fr`	80416	65225	186200	355846	432581	428024	653902	728661
`es-it`	80431	74633	216206	374894	469955	458432	713784	768111
`es-pt`	80173	80195	196733	340348	467287	454674	663178	711252
`fr-en`	65875	50411	146755	289242	345946	343148	551780	626083
`es-ca`	161127	159217	200983	217005	298134	289278	328737	329769
`es-gl`	24981	25284	25636	25761	30058	29736	29700	29482

Tabla 6.2: Evolución de los tamaños de los vocabularios de palabras y bipalabras para las diferentes políticas de alineamiento.



Bitexto	1:1		1:N	
	No Desp.	Desp.	Cont.	No Cont.
`de-en`	59,99 %	48,27 %	33,33 %	28,26 %
`en-es`	50,59 %	39,45 %	26,79 %	24,07 %
`es-fr`	46,29 %	39,77 %	27,56 %	23,60 %
`es-it`	46,54 %	40,82 %	30,12 %	26,89 %
`es-pt`	42,12 %	36,34 %	26,58 %	23,61 %
`fr-en`	54,03 %	42,45 %	31,63 %	29,21 %
`es-ca`	12,80 %	12,17 %	7,70 %	7,18 %
`es-gl`	5,76 %	5,74 %	5,09 %	5,05 %

Figura 6.8: Análisis de utilización de la palabra ϵ para las diferentes políticas de alineamiento.

La tabla 6.2 muestra la evolución del tamaño de los vocabularios de palabras y bipalabras para los diferentes pares de idiomas considerados y para las diferentes políticas de alineamiento propuestas. Las figuras 6.9, 6.10 y 6.11 muestran una representación de la tendencia que siguen los tamaños de los vocabularios de algunos pares de idiomas para cada una de las políticas de alineamiento consideradas. Cada una de las figuras contiene tres líneas representativas de los vocabularios de Σ_L (*roja*) y Σ_R (*verde*), así como una tercera (*azul*) que muestra la evolución del tamaño de Σ_B. En el eje `Y` de estas figuras se representa el número de símbolos (palabras o bipalabras), mientras que el eje[4] `X` muestra (de izquierda a derecha) las diferentes políticas de alineamientos: `1:1 No Desp.`, `1:1 Desp.`, `1:N Cont.` y `1:N No Cont.`

El bitexto `de-en` plantea un caso extremo en el análisis actual dada la enorme diferencia

[4]El rango de valores en este eje se comprende en $[0, 800000]$ para todos los pares de idiomas excepto para `(es,gl)`, cuyo rango se distribuye en $[0, 50000]$, dado el menor tamaño de este bitexto.

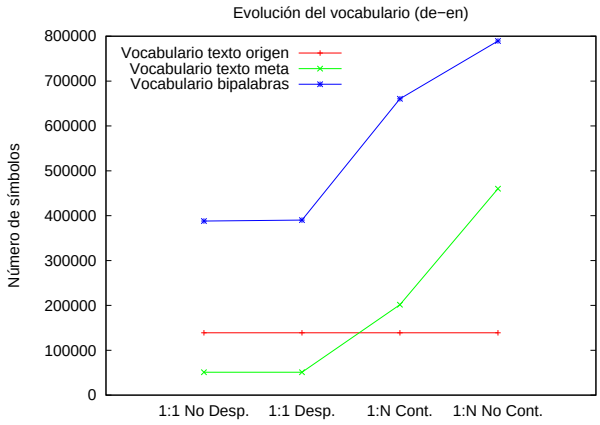


Figura 6.9: Evolución comparativa del tamaño de los vocabularios de palabras y bipalabras para el bitexto `de-en`.

gramatical existente entre los idiomas *alemán* e *inglés*. El primer síntoma de dicha diferencia queda patente en el tamaño de los vocabularios de palabras. En la tabla 6.2 puede observarse como $\sigma_L \approx 2,72\sigma_R$ (139010 palabras en alemán frente a las 51015 en inglés). Este resultado junto el elevado porcentaje de utilización de la palabra ϵ (interviene en un $59,99\,\%$ de los alineamientos para `1:1 No Desp.` y en un $48,28\,\%$ para `1:1 Desp.`) es indicativo de que las políticas de alineamiento *uno a uno* no se muestran como soluciones competitivas para el alineamiento de pares de idiomas diferentes. Consecuentemente, esto tendrá un impacto negativo en la efectividad obtenida para la compresión de estos bitextos debido a que la utilización masiva de ϵ aumenta el número de bipalabras necesarias para representar un determinado bitexto. Por lo tanto, ambos tipos de modelado se verán perjudicados por este motivo.

Sin embargo, la utilización de políticas *uno a muchos* mejora notablemente la calidad del alineamiento como puede verse en la tendencia descrita en la figura 6.8. La utilización de ϵ pasa a porcentajes del $33,33\,\%$ y del $28,26\,\%$, lo que permite reducir a más de la mitad el valor originalmente obtenido en las políticas *uno a uno*. Esto supone un importante aumento en el número de símbolos para los que se encuentra traducción en su idioma complementario lo que implica tanto una mejor calidad de la información representada como una reducción en el número total de símbolos que precisan ser modelados y codificados en el resultado comprimido. Sin embargo, como se observa en la figura 6.9, el coste de esta mejoría se paga en el importante crecimiento del número de palabras en Σ_R (inglés), cuyo tamaño es 9 veces mayor que el original (se pasa de 51015 a 460100 palabras). Esto penalizará significativamente el tamaño del resultado comprimido tanto por la necesidad de codificar una jerarquía de gran tamaño para la derivación de multipalabras (ver §6.4.1) como por el aumento de la longitud media de la palabra de código utilizada ante tal crecimiento en el número de palabras diferentes que precisan ser manejadas. Desde la perspectiva de los modelos orientados a bipalabras, el tamaño de Σ_B se duplica respecto al obtenido en `1:1` (de 387985 a 789381 bipalabras) con el impacto que esto supone desde la perspectiva anterior.

Los bitextos `en-es` y `es-pt`, tratados a continuación, se identifican en la segunda de las categorías anteriormente especificadas. Como puede observarse en la figura 6.10, la evolución de sus vocabularios describe una tendencia muy similar para las políticas de alineamiento consideradas. La diferencia básica radica en que $\sigma_L \approx \sigma_R$ para `es-pt`, mientras que $\sigma_L < \sigma_R$ para `en-es`. Esto influye en los valores absolutos que describen tanto la evolución de Σ_R como la de Σ_B. La principal diferencia entre estos pares de idiomas radica en el porcentaje de alineamientos en los que interviene la palabra ϵ.

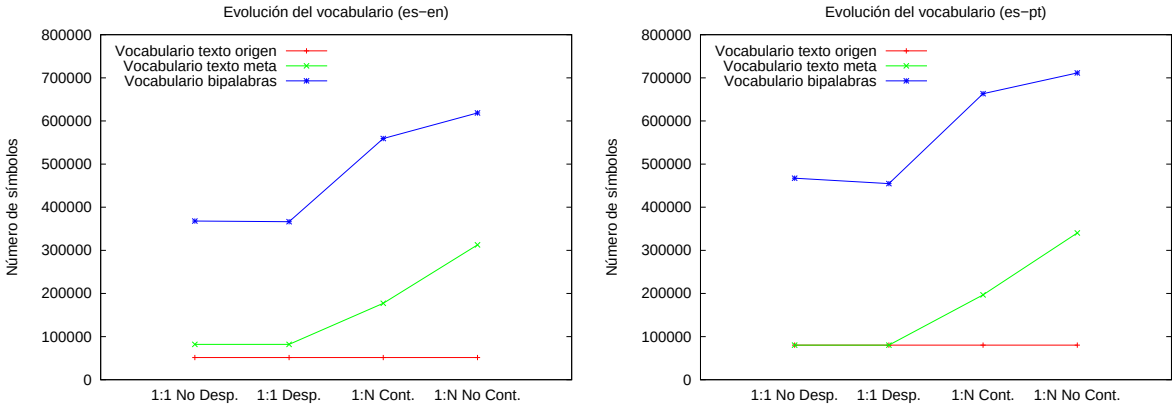


Figura 6.10: Evolución comparativa del tamaño de los vocabularios de palabras y bipalabras para los bitextos `en-es` y `es-pt`.

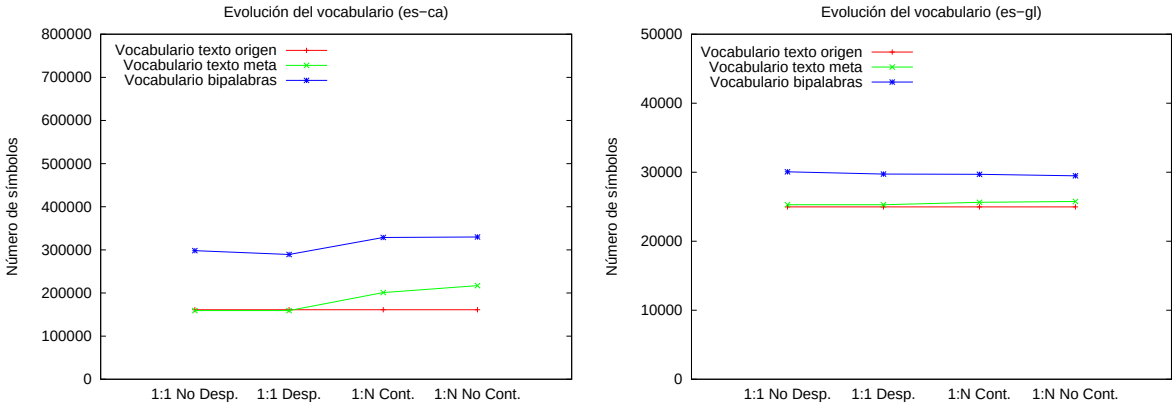


Figura 6.11: Evolución comparativa del tamaño de los vocabularios de palabras y bipalabras para los bitextos `es-ca` y `es-gl`.

Puede observarse en la figura 6.8 como el uso de ϵ en `en-es` se distribuye en el intervalo $[24,07\,\%,\ 50,59\,\%]$ mientras que en `es-pt` el valor máximo es ligeramente menor al anterior: $[23,61\,\%,\ 42,12\,\%]$. Esto supone que las políticas *uno a uno* obtienen representaciones de mejor calidad y con un menor número de alineamientos para `es-pt`; sin embargo, al considerar multipalabras, la representación de `en-es` mejora hasta alcanzar una tasa de uso de ϵ similar a la de `es-pt` lo que se traducirá en unas ratios de compresión comparables en ambos casos. Desde la perspectiva del modelado orientado a bipalabras, se observa como σ_B es mayor para `es-pt`.

Finalmente se estudian las propiedades estadísticas de los pares de idiomas altamente relacionados. Como puede observarse en la figura 6.8, el porcentaje de uso de ϵ es notablemente inferior a los valores obtenidos para los pares anteriores: mientras que para `es-gl` este valor apenas evoluciona en el intervalo $[5,05\,\%,\ 5,76\,\%]$, para el bitexto `es-ca` se sitúa en $[7,18\,\%,\ 12,80\,\%]$. Estos valores arrojan dos conclusiones claras:

- La calidad del alineamiento obtenido con las políticas *uno a uno* es significativamente mejor que para el resto de idiomas dado que apenas un $12,80\,\%$ de las palabras para `es-ca` y un $5,76\,\%$ para `es-gl` no están asociadas con su traducción en el idioma complementario.

- La utilización de multipalabras apenas mejora la calidad de la representación.

Esto supone, para este tipo de bitextos, que los modelos basados en bipalabras plantean soluciones más competitivas que para los casos anteriores, dado que el tamaño de su diccionario apenas se incrementa. Nótese como para `es-gl` el tamaño de los vocabularios se mantiene casi constante lo que demuestra, de nuevo, que el alineamiento *uno a uno* plantea un resultado competitivo para estos pares de idiomas. Por su parte, en `es-ca` sí se produce un ligero crecimiento de un $0,9\,\%$ pasando de 298134 a 329769 bipalabras.

Evolución del tamaño de la tabla de traducción. Denominamos *tabla de traducción* de un bitexto al conjunto de correspondencias identificadas entre las palabras de Σ_L y Σ_L. La tabla de traducción se corresponde con Σ_B en `2LCAB`, mientras que en `TRC` dicha tabla está distribuida en los diccionarios de traducción $\tau(p_L), \forall p_L \in \Sigma_L$.

La reducción en el porcentaje de uso de la palabra ϵ implica, desde la perspectiva actual, descubrir nuevas relaciones de traducción no identificadas anteriormente. Por lo tanto, las palabras previamente relacionadas con ϵ pasan a vincularse con una palabra determinada en el idioma complementario. Supongamos como ejemplo un bitexto `es-en` alineado con una política `1:1 No Desp.` En él, la palabra "*casa*" tiene tres ocurrencias pero ambas se representan con un alineamiento (*casa*, ϵ). De la misma manera, supongamos que las palabras "*house*" y "*home*" tampoco han podido ser alineadas con una palabra en castellano. Por lo tanto, se utilizarán dos alineamientos (ϵ, *house*) y (ϵ, *home*) para su representación. Sin embargo, es evidente que existe una relación de traducción mutua entre estas palabras y supongamos que ésta puede ser detectada sin más que pasar de una técnica `1:1 No Desp.` a `1:1 Desp.` Por lo tanto, dos de las ocurrencias de "*casa*" se alinean, respectivamente, con "*house*" y "*home*", mientras que su tercera ocurrencia sigue alineada con ϵ dado que no se ha podido establecer su relación de traducción. Además, supongamos que las palabras "*house*" y "*home*" tienen una segunda ocurrencia que, a su vez, se mantiene asociada con ϵ.

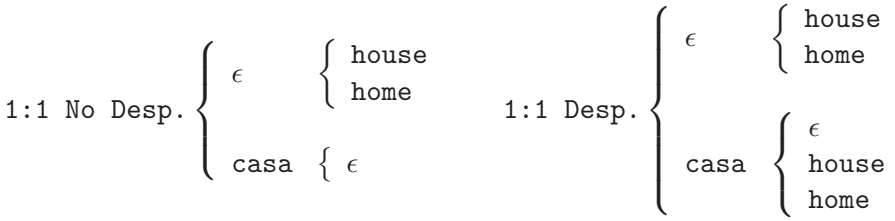


Figura 6.12: Configuración de un modelo orientados a palabras para el ejemplo de bitexto dado.

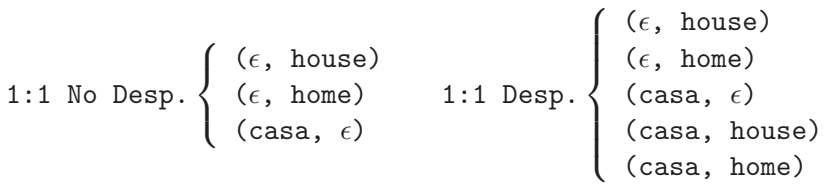


Figura 6.13: Configuración de un modelo orientados a bipalabras para el ejemplo de bitexto dado.

La figura 6.12 muestra la configuración de la tabla de traducción obtenida para el alineamiento `1:1 No Desp.` (izquierda) y para `1:1 Desp.` (derecha). Puede observarse que $\tau(\epsilon)$ mantiene dos elementos en el resultado obtenido por ambas políticas de alineamiento, mientras que $\tau($`casa`$)$ pasa de tener 1 a 3 elementos con la identificación de las nuevas relaciones de traducción. Este hecho representa que la mejora obtenida en la calidad del alineamiento tiende a aumentar el tamaño de la tabla de traducción al identificar nuevas relaciones. Desde la perspectiva de las

Bitexto	Origen (t_L)				Meta (t_R)			
	1:1		1:N		1:1		1:N	
	No Desp.	Desp.	Cont.	No Cont.	No Desp.	Desp.	Cont.	No Cont.
`de-en`	2,79	2,81	4,75	5,68	7,61	7,65	3,28	1,72
`en-es`	7,16	7,13	10,89	12,05	4,49	4,48	3,15	1,98
`es-fr`	5,38	5,32	8,13	9,06	6,63	6,56	3,51	2,05
`es-it`	5,84	5,70	8,87	9,55	6,30	6,14	3,30	2,05
`es-pt`	5,83	5,67	8,27	8,87	5,83	5,67	3,37	2,09
`fr-en`	5,25	5,21	8,38	9,50	6,86	6,81	3,76	2,16
`es-ca`	1,85	1,80	2,04	2,05	1,87	1,82	1,64	1,52
`es-gl`	1,20	1,19	1,19	1,18	1,19	1,18	1,16	1,14

Tabla 6.3: Evolución del número medio de traducciones asociadas con cada palabra en los textos origen y meta.

técnicas `TRC`, el número medio de traducciones de una palabra en el texto origen (t_L) determina el tamaño medio de sus vocabularios de traducción y, en términos globales, de la tabla de traducción utilizada para la compresión del bitexto. La tabla 6.3 muestra la evolución del número medio de traducciones por palabra en cada uno de los bitextos estudiados y para cada una de las políticas de alineamiento propuestas.

Puede observarse, en dicha tabla, como el valor de t_L decrece ligeramente al pasar de `1:1 No Desp.` a `1:1 Desp.` (excepto para `de-en`) aumentando progresivamente al considerar las políticas `1:N`. Esta evolución tiene dos consecuencias en `TRC`: 1) un incremento en el coste directamente asociado a la codificación de la tabla de traducción; y 2) un aumento en la longitud media de los códigos utilizados para la representación de cada palabra en su contexto en Σ_L.

Por su parte, la figura 6.13 muestra la configuración de Σ_B obtenida con las políticas de alineamiento `1:1 No Desp.` (izquierda) y `1:1 Desp.` (derecha), para el ejemplo anteriormente planteado. Puede observarse como el vocabulario inicial de 3 bipalabras pasa a tener 5 con la mejora obtenida en el alineamiento. Nótese, a su vez, que en este caso no sólo influye t_L sino que también lo hace el valor equivalente en el texto meta (t_R). La tabla 6.3 refleja como el valor t_R decrece con la mejora del alineamiento dado que se tiende a localizar, con mayor exactitud, las relaciones de traducción desde la perspectiva de $\mathcal{R}$. El efecto de t_L y t_R, en el tamaño de Σ_B, ha sido determinado empíricamente como: $\sigma_B = k(\log(t_L)\sigma_L + \log(t_R)\sigma_R)$, donde k depende de la relación existente entre el par de idiomas y, complementariamente, de política de alineamiento utilizada en cada caso para la obtención de la secuencia de bipalabras.

Como se observa en los resultados presentados en la tabla 6.2, el valor de σ_B crece con la mejora del alineamiento obtenido, tanto por la evolución de los valores t_L y t_R como por el incremento del tamaño de Σ_R. Esto supone, al igual que para `TRC`, que el coste de codificar la tabla de traducción en una representación orientada a bipalabras tiende a aumentar con la mejora de la calidad del alineamiento obtenido dado que el valor de σ_B crece de forma logarítmica respecto a t_L y de forma sublineal respecto al incremento de σ_R. Asimismo, la longitud media de palabra de código utilizada para la representación de cada bipalabra crece en la medida en la que se incrementa el valor de σ_B.

Evolución del tamaño de los bitextos. La diferentes políticas de alineamiento consideradas obtienen diferentes representaciones de la misma información contenida en el bitexto. Como se analizaba anteriormente, uno de los factores más importantes es la tasa de utilización de la palabra ϵ dado su efecto tanto en la calidad del alineamiento como en el tamaño de la tabla de traducción. Además, el valor ϵ influye también en el tamaño final de $\mathcal{B}$.

Supongamos un bitexto descrito sobre un conjunto de n bipalabras. Con independencia del tipo de modelo seleccionado, las técnicas propuestas en el capítulo actual lo representan con un número $O(n)$ de palabras de código: un compresor `TRC`, orientado a palabra, emitiría $2n$ códigos

para la compresión del bitexto mientras que 2LCAB emitiría la mitad para la representación del mismo bitexto. Sin embargo el valor de n, para un bitexto dado, no es un valor fijo sino que depende directamente de la opción de alineamiento seleccionada. Retomemos el bitexto de ejemplo utilizado al principio del capítulo:

(es) prefiero volver a la casa verde en que vivimos
(en) i would like to go back to the green house we live in

La utilización de una política `1:1 No Desp.` obtendría una representación compuesta por $n = 16$ bipalabras:

```
(,i) (,would) (prefiero,like) (,to) (volver,go) (,back) (a,to) (la,the)
(,green) (casa,house) (verde,) (,we) (,live) (en,in) (que,) (vivimos,)
```

Por su parte, utilizar una política `1:N No Cont.` representaría el bitexto anterior con $n = 10$ bipalabras:

```
(prefiero,i like,0,1) (,would) (volver,to go back) (a,to) (la,the)
(casa,house,1) (verde,green) (en,in,2) (que,) (vivimos, we live)
```

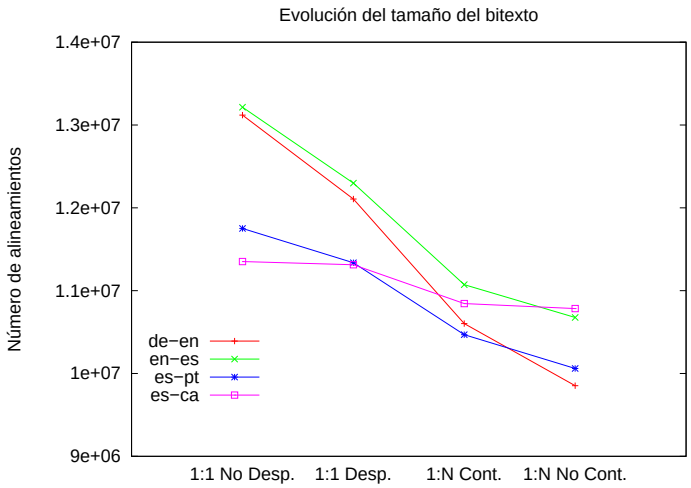
Esto supone que la misma información, incluida en el bitexto original, se representa con 6 bipalabras menos en el segundo caso. Esta reducción se obtiene a costa de aumentar, como se ha visto en el apartado anterior, el tamaño de la tabla de traducción con el que coste que esto tiene en su codificación. Por lo tanto, la efectividad de las técnicas de compresión planteadas radica en el compromiso obtenido entre el crecimiento de la tabla de traducción y la reducción del tamaño de la representación, resultados ambos de la mejora obtenida en la calidad del alineamiento.

La figura 6.14 muestra la evolución del tamaño de los bitextos considerados anteriormente de acuerdo a las diferentes políticas de alineamiento. El bitexto `es-gl` no se muestra en la gráfica dado que define un comportamiento prácticamente constante: su tamaño oscila entre 1063459 y 1069537 bipalabras.

Como era esperado, el número de bipalabras utilizadas en la representación del bitexto decrece al evolucionar de políticas *uno a uno* a las políticas *uno a muchos*. El bitexto `de-en` representa el caso más significativo como puede observarse en la figura 6.14. El número de bipalabras obtenidas en `1:N No Cont.` es un 33 % menor que las requeridas en `1:1 No Desp.` Por su parte, las reducciones obtenidas con `en-es` y `es-pt` son, respectivamente, de un 24 % y un 17 %, mientras que para el bitexto `es-ca` apenas se reduce un 5 %. Estos valores refuerzan las conclusiones anteriores, planteando las políticas *uno a uno* como una forma competitiva de representar textos de propiedades similares y requiriendo las *uno a muchos* para aquellos pares de idiomas más distantes entre sí.

6.5.2. Resultados experimentales

La sección actual detalla el estudio experimental llevado a cabo sobre los compresores propuestos con el fin de evaluar su efectividad en un entorno de explotación real. Para este proceso de experimentación se diseña un prototipo basado en 2LCAB que integra un código canónico de `Huffman` como técnica de codificación (el compresor resultante se refiere como $\texttt{2LCAB}_{huff}$) con el fin de comparar la efectividad de los modelos de representación del bitexto en igualdad de condiciones dado que las técnicas `TRC` utilización codificación de `Huffman` orientada a bit. Por su parte, la implementación de 2LCAB utilizada en la sección siguiente para el análisis de sus propiedades de búsqueda sigue la propuesta original [ABMPSM09] (referida como $\texttt{2LCAB}_{etdc}$) codificando la secuencia de bipalabras con una técnica orientada a byte (`ETDC`).



Bitexto	1:1		1:N	
	No Desp.	Desp.	Cont.	No Cont.
`de-en`	13120052	12106577	10602529	9852297
`en-es`	13214990	12298572	11072964	10676325
`es-fr`	12487356	11979311	10747408	10189344
`es-it`	12086365	11652137	10642364	10171300
`es-pt`	11751300	11336391	10468679	10060951
`fr-en`	13378814	12394893	11344378	10955977
`es-ca`	11351053	11312923	10843650	10783467
`es-gl`	1069537	1069390	1063950	1063459

Figura 6.14: Evolución del tamaño del bitexto (en número total de alineamientos) en distintos pares de idiomas y con las diferentes políticas de alineamiento.

El presente estudio experimental analiza tanto la efectividad de nuestras técnicas de compresión como su eficiencia temporal tanto en compresión como en descompresión. En primer lugar nos centraremos en una comparativa de las ratios de compresión obtenidas. Este estudio se lleva a cabo respecto a la efectividad obtenida por diferentes técnicas universales de compresión debido a que no se han podido localizar las fuentes de ninguna de las técnicas reseñadas en la sección de trabajo relacionado del presente capítulo (ver §6.1).

Los compresores universales se eligen, al igual que en los capítulos anteriores, con el objetivo de cubrir de la forma más completa posible las diferentes técnicas de compresión existentes. Por un lado se utiliza `ppmdi` como representante de una técnica de orden superior de carácter predictivo; `gzip` y `p7zip` se seleccionan como técnicas basadas en diccionario sobre diferentes variantes de `LZ` y, finalmente, `bzip2` combina el preprocesamiento obtenido por la transformada de *Burrows-Wheeler* y su posterior compresión.

La tabla 6.4 recoge los resultados obtenidos con estos compresores para cada uno de los pares de idiomas anteriormente utilizados. Puede observarse como `ppmdi` representa, de forma generalizada, la opción más efectiva salvo en los pares de idiomas más cercanos: `es-ca` y `es-gl`, donde `p7zip` obtiene unas mejores tasas de compresión. Todos aquellos bitextos para los que `ppmdi` representa la mejor opción se comprimen con ratios próximas al 20 %. Esto supone 1 punto porcentual de mejora respecto a `bzip2` y 2 respecto a `p7zip` (como puede observarse en la tabla, la efectividad de `gzip` no es competitiva respecto a las anteriores). La uniformidad de estos resultados demuestra que la relación entre los idiomas no plantea ningún tipo de regularidad genérica que pueda ser detectada por las técnicas universales excepto en el bitexto `es-gl`. En este caso, ambos idiomas no sólo se caracterizan por utilizar palabras derivadas de una raíz

Bitexto	bzip2	gzip	p7zip	ppmdi
de-en	22,19 %	31,59 %	21,61 %	**20,35 %**
en-es	22,01 %	31,38 %	21,39 %	**20,12 %**
es-fr	21,51 %	30,80 %	20,75 %	**19,57 %**
es-it	21,88 %	31,00 %	21,15 %	**19,98 %**
es-pt	21,94 %	31,10 %	21,11 %	**20,02 %**
fr-en	21,65 %	31,10 %	21,06 %	**19,76 %**
es-ca	27,21 %	37,09 %	**24,41 %**	25,43 %
es-gl	13,08 %	20,84 %	**10,62 %**	11,33 %

Tabla 6.4: Tasas de compresión obtenidas por diferentes compresores universales.

2V	
Bitexto	1:1
	No Desp.
de-en	24,66 %
en-es	24,94 %
es-fr	24,68 %
es-it	24,81 %
es-pt	24,46 %
fr-en	25,15 %
es-ca	27,53 %
es-gl	23,04 %

$\texttt{TRC}_{(0;1)}$				
Bitexto	1:1		1:N	
	No Desp.	Desp.	Cont.	No Cont.
de-en	20,34 %	**20,02 %**	20,04 %	20,68 %
en-es	19,54 %	18,77 %	**18,58 %**	18,90 %
es-fr	19,01 %	18,55 %	**18,49 %**	18,85 %
es-it	19,14 %	**18,79 %**	18,94 %	19,37 %
es-pt	18,37 %	**17,99 %**	18,09 %	18,49 %
fr-en	20,15 %	19,25 %	**19,08 %**	19,40 %
es-ca	16,89 %	16,83 %	**16,57 %**	16,61 %
es-gl	14,16 %	**14,15 %**	14,18 %	14,19 %

Tabla 6.5: Efectividad de las técnicas 2V y $\texttt{TRC}_{(0;1)}$.

común sino que, en muchos casos, las palabras son similares y/o tienen morfemas comunes lo cual facilita su compresión. Esto explica, a su vez, porque p7zip mejora a ppmdi, dado que el algoritmo lzma en el que se basa es capaz de detectar, como frases del diccionario, cada una de estas raíces comunes y, por tanto, optimizar su representación con un menor número de bits al requerido en un algoritmo predictivo como ppdmi. Finalmente, los resultados obtenidos para es-ca plantean una contradicción dado que, a pesar de la relación de proximidad existente entre ellos, son siempre peores a los obtenidos para el resto de pares de idiomas bajo las propiedades de compresión asociadas a cada una de las técnicas utilizadas. No se ha encontrado una explicación concreta que justifique esos resultados.

A continuación se evalúa la efectividad de las técnicas TRC y $\texttt{2LCAB}_{huff}$. La tabla 6.5 muestra los resultados obtenidos por 2V y $\texttt{TRC}_{(0;1)}$. La efectividad de 2V (tabla izquierda) es significativamente peor que la obtenida por $\texttt{TRC}_{(0;1)}$ para los mismos textos y con la misma política de alineamiento 1:1 No Desp. La diferencia está comprendida entre 4 y 11 puntos porcentuales en favor de $\texttt{TRC}_{(0;1)}$ lo cual demuestra cómo las técnicas genéricas orientadas a palabras para lenguaje natural no son capaces de identificar la redundancia semántica inherente al bitexto y su efectividad es comparable a la obtenida al comprimir un texto genérico en lenguaje natural (nótese como los resultados obtenidos por 2V son coherentes con las conclusiones reportadas por Turpin y Moffat [TM97] acerca de que una codificación de Huffman orientada a palabras obtiene tasas de compresión de, aproximadamente, el 25 %). Los resultados obtenidos por 2V con otras técnicas de alineamiento se descartan dado que, en todos los casos, son peores que los obtenidos con 1:1 No Desp.

La misma tabla 6.5 (derecha) contiene los resultados obtenidos con $\texttt{TRC}_{(0;1)}$. Su análisis se lleva a cabo de acuerdo a la categorización de pares de idiomas planteada en el apartado anterior. Las tasas de compresión obtenidas para de-en están comprendidas en el rango [20, 02 %, 20, 68 %], lo cual supone una ligera mejoría (en el mejor caso) frente al 20, 35 % obtenido por ppmdi. Los mejores resultados se obtienen para las políticas de alineamiento 1:1 Desp. y 1:N Cont. con un 20, 02 % y un 20, 04 %, respectivamente. Esta similitud en la compresión demuestra la capacidad del modelado TRC para sobreponerse al gran crecimiento de Σ_R (de 51015 a 201511 símbolos) y al incremento de t_L (que pasa de 2, 81 a 4, 75 traducciones/palabra).

Bitexto	1:1		1:N	
	No Desp.	Desp.	Cont.	No Cont.
de-en	18,42 %	**18,23 %**	18,24 %	18,91 %
en-es	17,32 %	16,64 %	**16,48 %**	16,78 %
es-fr	16,74 %	16,43 %	**16,39 %**	16,75 %
es-it	16,82 %	**16,61 %**	16,79 %	17,22 %
es-pt	16,06 %	**15,83 %**	15,96 %	16,36 %
fr-en	17,22 %	16,47 %	**16,36 %**	16,68 %
es-ca	14,85 %	14,99 %	**14,75 %**	14,78 %
es-gl	**10,44 %**	10,50 %	10,54 %	10,54 %

Tabla 6.6: Efectividad de la técnica $\texttt{TRC}_{(1;1)}$.

Finalmente, la compresión obtenida con `1:N No Cont.` empeora ligeramente hasta un $20,68\,\%$ debido, principalmente, al coste asociado a la codificación de la tabla de traducción cuyo tamaño comprimido es más de 3 veces superior al requerido en `1:1 Desp.` Esto supone que el modelo `TRC` es capaz de mantener unas tasas de compresión estables para `de,en` con las diferentes políticas de alineamiento consideradas.

Los pares de idiomas `en-es`, `es,fr`, `es-it`, `es-pt` y `fr-en` obtienen unos resultados comparables, aunque los mejores son los obtenidos para idiomas más próximos. Las tasas de compresión más bajas se consiguen para el bitexto `es-pt` y se localizan en el intervalo $[17,99\,\%,\ 18,37\,\%]$. La mejora, respecto al $20,02\,\%$ obtenido por `ppmdi`, ya supera los 2 puntos porcentuales siendo las políticas `1:1 Desp.` y `1:N Cont.` las que mejor efectividad consiguen. La explicación es similar a la dada para `de-en` con la diferencia de que en este caso el número medio de traducciones (t_L) llega a $8,27$ traducciones/palabras mientras que la proporción en la que se incrementa el tamaño de Σ_R es menor (de 80195 a 196733). Por otro lado, las tasas de compresión obtenidas para `f-,en` son las más altas, en este subconjunto de idiomas, y se distribuyen en el intervalo $[19,08\,\%,\ 20,15\,\%]$. En este caso la mejoría frente a `ppmdi` es menor considerando la ratio de $19,76\,\%$ que ésta obtiene.

Finalmente, los pares de idiomas más relacionados son los que obtienen una mejor efectividad. Por un lado, `es-gl` distribuye sus resultados en $[14,15\,\%\ ,14,19\,\%]$ lo que supone que las diferentes políticas de alineamiento apenas afectan al resultado de la compresión. En este caso, $\texttt{TRC}_{(0;1)}$ es menos efectiva que `ppmdi` o `p7zip` que, como se indicaba anteriormente, son capaces de aprovechar la gran similitud existente entre las palabras que forman los vocabularios de ambos idiomas. Por su parte, las tasas de compresión obtenidas para `es-ca` también suponen una mayor efectividad que la obtenida para el resto de idiomas. En este caso, sus resultados se distribuyen en el rango $[16,57\,\%,\ 16,89\,\%]$ lo que supone mejorar hasta en 8 puntos porcentuales el mejor resultado obtenido por una técnica universal (en este caso `p7zip`). La mejor efectividad se consigue para `1:N Cont.` debido a la gran reducción obtenida en la utilización de ϵ cuyo porcentaje pasa del $12,17\,\%$ al $7,70\,\%$.

La tabla 6.6 muestra los resultados obtenidos por $\texttt{TRC}_{(1;1)}$. Como se explicaba anteriormente, el cambio de esta técnica respecto a $\texttt{TRC}_{(0;1)}$ radica, exclusivamente, en la compresión de $\mathcal{L}$. Mientras que en el caso anterior se utiliza una variación de un `Huffman` de palabras, $\texttt{TRC}_{(1;1)}$ adapta el compresor $\texttt{E-G}_1$, presentado en el capítulo previo, de tal forma que la representación obtenida por su grafo de palabras no se vea perjudicada por la utilización de la palabra ϵ (ver §6.4.3). Los datos mostrados en este tabla se obtienen parametrizando $\texttt{E-G}_1$ con $\alpha = 2^{10}$ y utilizando una política de organización/reemplazo `LFU`.

La evolución de los resultados para cada uno de los pares de idiomas y para cada política de alineamiento es similar a la comentada en el apartado anterior, dado que la diferencia únicamente radica en la compresión del texto origen. Los resultados actuales son entre 2 y 4 puntos porcentuales mejores que los presentados para $\texttt{TRC}_{(0;1)}$, lo cual supone una mejora aún mayor frente a los compresores universales. Nótese cómo $\texttt{TRC}_{(1;1)}$ consigue mejorar la compresión obte-

Bitexto	1:1		1:N	
	No Desp.	Desp.	Cont.	No Cont.
de-en	19,98 %	**19,83 %**	20,91 %	22,68 %
en-es	19,29 %	**18,68 %**	19,25 %	20,19 %
es-fr	18,89 %	**18,50 %**	19,34 %	20,54 %
es-it	19,18 %	**18,87 %**	20,02 %	21,25 %
es-pt	18,46 %	**18,09 %**	19,05 %	20,16 %
fr-en	19,75 %	**19,03 %**	19,68 %	20,71 %
es-ca	16,69 %	16,61 %	**16,59 %**	16,70 %
es-gl	13,77 %	**13,75 %**	13,81 %	13,82 %

Tabla 6.7: Efectividad de la técnica $\texttt{2LCAB}_{huff}$.

nida por las técnicas universales en el único caso en el que $\texttt{TRC}_{(0;1)}$ no es superior. Mientras que `p7zip` obtiene una tasa de de compresión del $10,62\,\%$ para el bitexto `es-gl`, $\texttt{TRC}_{(1;1)}$ consigue distribuir sus ratios de compresión en el intervalo $[10,44\,\%,\ 10,54\,\%]$.

Finalmente, se estudian los resultados obtenidos sobre el modelo orientado a bipalabras que implementa `2LCAB`. La comparación, respecto a un modelo de palabras, se hace frente a $\texttt{TRC}_{(0;1)}$, dado que en ambos casos se utilizan modelos de orden 0 para la representación del bitexto sobre diferentes alfabetos de entrada.

La tabla 6.7 muestra las tasas de compresión obtenidas por $\texttt{2LCAB}_{huff}$. Puede observarse como los mejores resultados se obtienen para alineamientos *uno a uno* y, por tanto, en aquellas políticas que no consideran el uso de multipalabras en $\mathcal{R}$. Este resultado es esperable de acuerdo al análisis anterior acerca del incremento del tamaño de Σ_B en políticas de alineamiento *uno a muchos*. Sin embargo, `2LCAB` mantiene la posibilidad de acceder al bitexto a partir de cualquiera de los idiomas que lo forman mientras que una técnica `TRC` se podría adaptar para propósitos de búsqueda aunque ésta sólo podría llevarse a cabo a través de $\mathcal{L}$.

A continuación se muestra un resumen de la eficiencia que presentan las técnicas anteriores en sus procesos de compresión y descompresión. En primer lugar se estudia el impacto asociado a manejar, en la técnica de compresión, las diferentes políticas de alineamiento propuestas. Dicho análisis se lleva a cabo sobre bitextos `en-es` y `es-ca` y sus procesos de compresión y descompresión con $\texttt{TRC}_{(0;1)}$. Para cada bitexto se consideran tamaños 1, 10 y 100MB. con el fin de analizar la evolución de la eficiencia de las técnicas para diferentes tamaños de bitexto. La figura 6.15 plantea los tiempos de compresión y descompresión obtenidos con cada una de las políticas de alineamiento propuestas.

Las dos gráficas superiores representan los tiempos obtenidos para el bitexto `en-es`. Puede observarse como el procesamiento de alineamientos *uno a muchos* aumenta ligeramente (entre un $5\,\%$ y un $10\,\%$) los tiempos de compresión mientras que el efecto es más marcado en el proceso de descompresión debido a la necesidad de decodificar la jerarquía utilizada para representar las multipalabras. La utilización de alineamientos *uno a muchos* aumenta hasta un $30\,\%$ el tiempo de descompresión respecto al obtenido con las políticas *uno a uno*. La tendencia es muy similar para el bitexto `es-ca`, aunque el efecto es menos significativo tanto en compresión como en descompresión debido al menor número de multipalabras consideradas para la representación del bitexto. Los tiempos de compresión y descompresión que presentan el resto de técnicas siguen una tendencia similar a la mostrada por $\texttt{TRC}_{(0;1)}$. Por lo tanto, se concluye que el manejo de multipalabras hace más lentos los procesos de compresión y descompresión, siendo en este último caso en el que los tiempos crecen en una mayor proporción.

Las tablas 6.8 y 6.9 plantean una comparativa entre los tiempos de compresión/descompresión obtenidos al utilizar técnicas universales para la compresión de los bitextos `en-es` y `es-ca` y los requeridos tanto por las técnicas `TRC` como por $\texttt{2LCAB}_{huff}$. Para estos casos se consideran los tiempos obtenidos sobre la política de alineamiento `1:1 No Desp.`

`gzip` obtiene los mejores tiempos de compresión y descompresión, aunque su efectividad

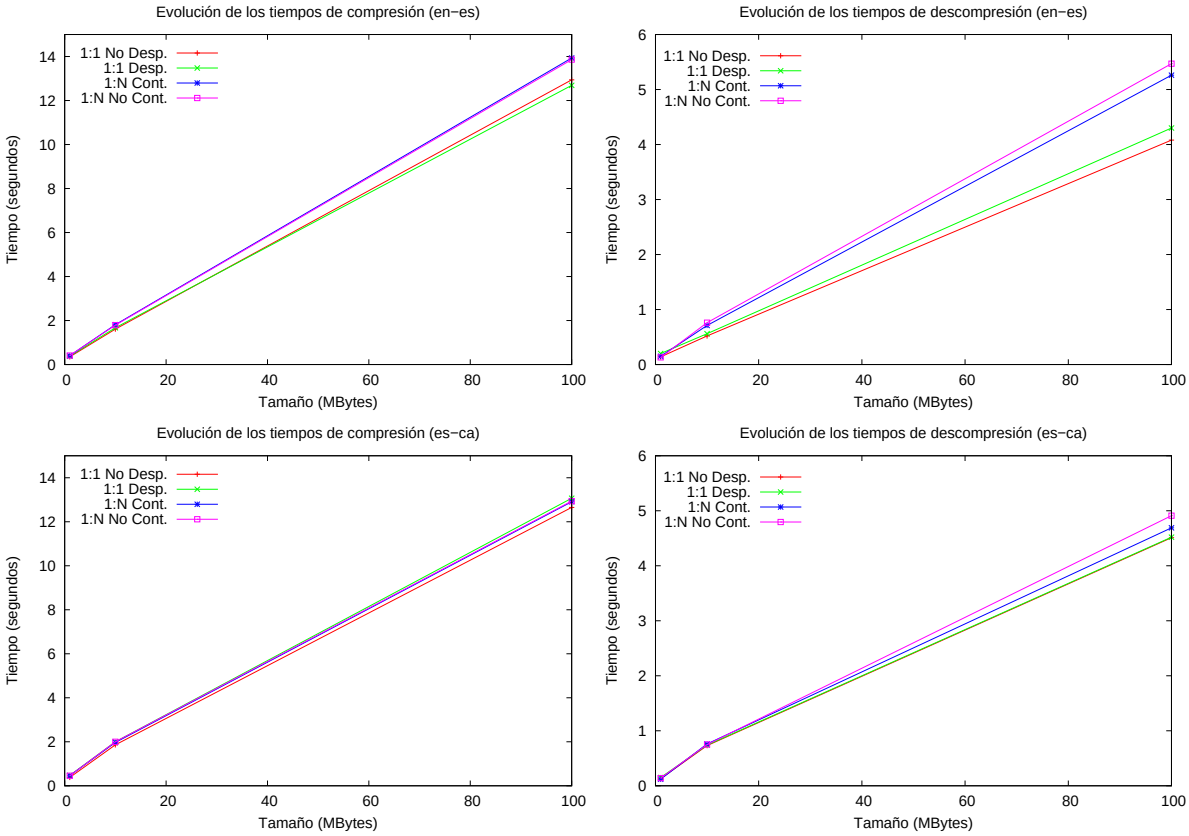


Figura 6.15: Tiempos de compresión/descompresión para (`en,es`) (arriba) y (`es,ca`) (abajo).

Bitexto	gzip	p7zip	bzip2	ppmdi	$\text{TRC}_{0,1}$	$\text{TRC}_{1,1}$	2LCAB_{huff}
en-es	3,81	143,89	15,20	26,88	12,84	18,17	8,72
es-ca	4,34	146,91	15,43	31,58	12,95	18,77	8,81

Tabla 6.8: Comparativa de tiempos de compresión (en segundos) para `en-es` y `es-ca`.

Bitexto	gzip	p7zip	bzip2	ppmdi	$\text{TRC}_{0,1}$	$\text{TRC}_{1,1}$	2LCAB_{huff}
en-es	1,03	2,23	6,32	27,26	4,08	7,35	2,66
es-ca	1,06	2,43	6,90	32,35	4,51	8,11	3,10

Tabla 6.9: Comparativa de tiempos de descompresión (en segundos) para `en-es` y `es-ca`.

no es competitiva en el entorno actual como se ha planteado previamente. De entre todas las técnicas propuestas, 2LCAB_{huff} representa la opción más eficiente. Este es un resultado evidente considerando que `2LCAB` lleva a cabo un procesamiento del bitexto basado en bipalabras y, por tanto en tiempo $O(n)$, mientras que `TRC` requiere un tiempo $O(2n)$ de acuerdo a su modelado basado en palabras. Nótese que $\text{TRC}_{(1;1)}$ requiere los tiempos más altos de compresión y descompresión viéndose penalizada por la utilización de un modelo de orden 1 para la representación y codificación de $\mathcal{L}$. Aún así, nuestras técnicas muestran, en términos generales, una mayor eficiencia que las universales. Sólo `bzip2` es capaz de mejorar ligeramente a $\text{TRC}_{(1;1)}$, pero tanto `p7zip` como `ppmdi` obtienen tiempos de compresión notablemente peores. En lo que respecta a descompresión, 2LCAB_{huff} obtiene tiempos competitivos incluso con `p7zip` cuya característica principal radica en el desarrollo de procesos de descompresión altamente veloces. Por su parte las técnicas `TRC` presentan unos tiempos de descompresión comparables a los de `bzip2` y, en todos los casos, notablemente mejores que los obtenidos por `ppmdi` (entre 4 y 8 veces) cuyas

Bitexto	2LCAB$_{huff}$	2LCAB$_{etdc}$	2V
de-en	20,34 %	22,66 %	32,88 %
en-es	19,54 %	21,86 %	32,77 %
es-fr	19,01 %	21,30 %	31,44 %
es-it	19,14 %	21,69 %	31,45 %
es-pt	18,37 %	20,87 %	30,76 %
fr-en	20,15 %	22,37 %	33,22 %
es-ca	16,89 %	19,22 %	32,39 %
es-gl	14,16 %	16,74 %	28,55 %

Tabla 6.10: Comparación de la efectividad obtenida por la técnica 2LCAB sobre codificación de Huffman canónica (2LCAB$_{huff}$) y ETDC (2LCAB$_{etdc}$) y 2V sobre ETDC.

Bitexto	Compresión		Descompresión	
	2LCAB$_{huff}$	2LCAB$_{etdc}$	2LCAB$_{huff}$	2LCAB$_{etdc}$
en-es	8,72	7,41	2,66	1,34
es-ca	8,81	7,36	3,10	1,95

Tabla 6.11: Comparativa de tiempos de compresión/descompresión (en segundos) para en-es y es-ca.

tasas de compresión son (de acuerdo a lo indicado en la tabla 6.4) las más competitivas en el contexto actual.

6.5.3. Variantes de búsqueda sobre el bitexto comprimido

La sección actual se centra en el análisis de la técnica 2LCAB$_{etdc}$ obtenida a partir de un modelado basado en bipalabras y una codificación orientada a byte sobre ETDC. Nótese que la presente experimentación se ejecuta sobre alineamiento 1:1 No Desp. que es la alternativa que actualmente soporta las operaciones de búsqueda analizadas. Todas las operaciones se llevan a cabo sobre el bitexto en-es y los tiempos se obtienen sobre un proceso compuesto por 5 réplicas independientes de cada experimento.

En este caso debe considerarse que ETDC plantea la técnica orientada a palabras más rápida para llevar a cabo búsqueda sobre texto comprimido [BFNP07] lo cual favorece la eficiencia obtenida por 2LCAB$_{etdc}$. Asimismo, y con el objetivo de obtener una comparación más completa, se considera la implementación de un prototipo de prueba basado en 2V (ver el *baseline* propuesto en la Sección §6.3.1) sobre codificación ETDC. Las operaciones de búsqueda en 2V se ejecutan sobre el mismo algoritmo Set-Horspool (ver §3.2.2) utilizado en 2LCAB$_{etdc}$.

La tabla 6.10 muestras las tasas de compresión obtenidas al conjuntar un modelo 2LCAB orientado a bipalabras con sendas técnicas de codificación orientadas a bit (2LCAB$_{huff}$) y byte (2LCAB$_{etdc}$). Los resultados mostrados en esta tabla se utilizan para evaluar el coste de pasar de una representación orientada a bit a su equivalente orientada a byte, considerando que las operaciones de búsqueda a nivel de byte son notablemente más eficientes. Finalmente, la columna titulada 2V plantea las tasas de compresión obtenidas por el prototipo anteriormente comentado.

Estos resultados muestran como el coste de pasar a una representación orientada a byte es apenas de $2-3$ puntos porcentuales para todos los pares de idiomas considerados. Por su parte, 2LCAB$_{etdc}$ mejora las tasas de compresión obtenidas por 2V entre $10-12$ puntos porcentuales con el consiguiente ahorro en espacio que esto conlleva e, indirectamente, con la mejoría que esto supone en la efectividad obtenida por las operaciones de búsqueda que se tratan a continuación.

Finalmente, la tabla 6.11 compara los tiempos de compresión y descompresión obtenidos por 2LCAB$_{huff}$ y 2LCAB$_{etdc}$ sobre los bitextos en-es y es-ca. Puede observarse como la versión orientada a byte (2LCAB$_{etdc}$) es siempre más eficiente dado que se beneficia de la mayor velocidad de procesamiento asociada al uso de una codificación ETDC respecto a la codificación de Huffman.

Búsqueda de traducciones. La búsqueda de traducciones, especificada en el algoritmo 6.4, utiliza la jerarquía de vocabularios (en dos niveles) para responder a esta consulta sin necesidad de acceder al bitexto comprimido. Por su parte, `2V` necesita recorrer completamente el bitexto comprimido, localizar todas las ocurrencias de la palabra solicitada y, de acuerdo con ello, obtener todas las palabras con las que ésta se relaciona en el idioma complementario. A partir de este resultado puede seleccionar el conjunto de palabras diferentes que conforman las traducciones para la palabra dada.

Categoría	Ocurrencias	2LCAB	2V
$\texttt{Trads}_{alta}$	95, 95	0, 0042 (0, 000044)	0, 1346 (0, 001838)
$\texttt{Trads}_{media}$	5, 58	0, 0043 (0, 000762)	0, 1340 (0, 023264)
$\texttt{Trads}_{baja}$	1, 01	0, 0046 (0, 004542)	0, 1341 (0, 117632)

Tabla 6.12: Tiempos de búsqueda de traducciones (en segundos) en el bitexto `en-es`.

La tabla 6.12 muestra los resultados obtenidos en los procesos de búsqueda de traducciones. La columna *categoría* refiere cada una de las tres categorías de búsqueda, confeccionadas por 100 palabras seleccionadas aleatoriamente. La columna *traducciones* indica el número medio de traducciones asociadas a cada categoría. Esto es, la categoría $\texttt{Trads}_{alta}$ está formada por palabras en inglés con un alto número de traducciones en castellano: 95, 95. De la misma, las palabras de $\texttt{Trads}_{media}$ se relacionan con 5, 58 palabras en castellano, mientras que $\texttt{Trads}_{baja}$ está formada por palabras que se traducen por 1, 01 palabra en castellano. Las columnas `2LCAB` y `2V` muestran el tiempo total requerido para llevar a cabo el proceso de búsqueda y, entre paréntesis, el tiempo medio necesario para localizar cada una de las traducciones asociadas a cada palabra en cada una de las categorías. Esto supone que el tiempo total de la operación es prácticamente similar para cada una de las categorías (0, 0042, 0, 0043 y 0, 0046 segundos, respectivamente, para $\texttt{Trads}_{alta}$, $\texttt{Trads}_{media}$ y $\texttt{Trads}_{baja}$) dado que este coste está principalmente asociado al recorrido del diccionario de bipalabras. Esto supone que el número de resultados no tiene una incidencia significativa en el rendimiento de la operación. Por lo tanto, cuánto mayor sea el número de traducciones menor será el coste medio asociado a la localización de cada una de ellas. Esto explica que el coste medio por traducción sea de 44 microsegundos para $\texttt{Trads}_{alta}$, $762\mu s$ para $\texttt{Trads}_{media}$ y $4542\mu s$ $\texttt{Trads}_{baja}$.

Por su parte, el rendimiento mostrado por `2V` no es competitivo para esta operación dado que paga el coste de recorrer completamente el bitexto comprimido. Por lo tanto, el coste global comprendido en [0, 1340 0, 1346] segundos es más de 30 veces superior al obtenido al buscar directamente sobre el diccionario de bipalabras comprimido y es comparable al mostrado a continuación para las operaciones de búsqueda de palabras y bipalabras.

Búsqueda de palabras/bipalabras. Las búsquedas de palabras y bipalabras representan operaciones semánticamente diferentes aunque su implementación, desde una perspectiva algorítmica, se lleva a cabo sobre las mismas operaciones en `2LCAB`. Dada una búsqueda de palabras, en primer lugar se identifican (mediante la búsqueda de traducciones anterior) el código identificador de las diferentes bipalabras en las que interviene la palabra solicitada. Si la búsqueda es de bipalabras, únicamente se recuperará un código en la búsqueda de traducciones: el asociado a la bipalabra solicitada. Este conjunto de códigos se utilizan posteriormente como entrada al algoritmo `Set-Horspool` que localiza las ocurrencia de la palabra en el bitexto comprimido. Por su parte, el proceso en `2V` es similar al comentado para la búsqueda de traducciones. Esto supone buscar con `Set-Horspool` el código de la palabra solicitada.

Para esta operación se construyen, nuevamente, tres categorías de búsqueda formada por 100 palabras diferentes caracterizadas, respectivamente, por una frecuencia de aparición alta, media y baja ($\texttt{Ocs}_{alta}$, $\texttt{Ocs}_{media}$ y $\texttt{Ocs}_{baja}$). La tabla 6.13 muestra los resultados asociados con esta

Categoría	Ocurrencias	2LCAB	2V
$\texttt{Ocs}_{alta}$	12629, 55	0, 0964 (0, 000008)	0, 1351 (0, 000011)
$\texttt{Ocs}_{media}$	53, 92	0, 0341 (0, 000632)	0, 1344 (0, 002493)
$\texttt{Ocs}_{baja}$	1, 25	0, 0295 (0, 023575)	0, 1342 (0, 107360)

Tabla 6.13: Tiempos de búsqueda de palabras/bipalabras en el bitexto (en,es).

Categoría	Snippets	2LCAB
$\texttt{Ocs}_{alta}$	12629, 55	0, 1832 (0, 000015)
$\texttt{Ocs}_{media}$	53, 92	0, 0360 (0, 000668)
$\texttt{Ocs}_{baja}$	1, 25	0, 0323 (0, 025847)

Tabla 6.14: Tiempos de extracción de *snippets* en el bitexto en-es.

operación. La categoría $\texttt{Ocs}_{alta}$ está formada por palabras con un número medio de 12629, 55 ocurrencias en el bitexto. El tiempo medio de ejecución es de $0, 0964s$ (nótese que $0, 0042s$ corresponden a la identificación de las traducciones de la palabra) lo que supone que localizar cada ocurrencia de la palabra tiene un coste medio de $8\mu s$. Para $\texttt{Ocs}_{media}$ se requiere un tiempo medio de ejecución de $0, 0341s$ lo que implica un coste de $632\mu s$ por ocurrencia. Finalmente, para aquellas palabras con una baja frecuencia de aparición, el coste de localizar cada ocurrencia crece notablemente hasta los $23575\mu s$. Por su parte, el coste de la operación en 2V es similar al obtenido para el caso anterior y, nuevamente, menos competitivo que el conseguido por 2LCAB, aunque para aquellas palabras con muchas ocurrencias se aproxima notablemente alcanzando un tiempo medio de $11\mu s$ por ocurrencia.

Extracción de *snippets*. Finalmente se estudia el coste de extracción de *snippets* en 2LCAB. Para el análisis de esta operación se utilizan las configuraciones de palabras consideradas en la operación anterior. Los *snippets* extraídos están formados por 11 palabras: el resultado localizado y las 5 palabras previas y posteriores al mismo. Nótese que los tiempos calculados incluyen la extracción del *snippet* tanto en el idioma en el que se expresa la consulta como en su complementario.

Los tiempos mostrados en la columna 2LCAB, de la tabla 6.14, comprenden tanto el coste de localizar el resultado como el tiempo necesario para extraer el conjunto de palabras que lo circundan. Por lo tanto, si nos fijamos en el tiempo mostrado para $\texttt{Ocs}_{alta}$, de los $0, 1832s$ necesarios para completar la operación, $0, 0964s$ corresponden a la localización de las ocurrencias y $0, 0868s$ se utilizan en la extracción del *snippet*. Esto supone que para las palabras comprendidas en este categoría, se requiere $15\mu s$ para la localización y extracción de *snippet* asociado a cada ocurrencia. Para el resto de categorías, el tiempo promedio por ocurrencia está fuertemente ligado al tiempo de localización y se cifra en $668\mu s$ y $25847\mu s$, respectivamente, para $\texttt{Ocs}_{media}$ y $\texttt{Ocs}_{baja}$.

6.6 Conclusiones y trabajo futuro

El crecimiento masivo de la información disponible en múltiples idiomas ha propiciado el desarrollo y asentamiento de numerosos tipos de aplicaciones tanto monolingües como de carácter multilingüe. Los *bitextos* plantean una herramienta básica en muchas de estas aplicaciones dado que aportan un conocimiento lingüístico esencial para su funcionamiento. En resumen, un bitexto contiene la misma información expresada en dos idiomas diferentes lo cual permite interpretar cada una de las versiones de la misma información como una anotación del significado de la complementaria. Esto supone que la información contenida en el bitexto está caracterizada por una gran cantidad de redundancia semántica inherente a su propia naturaleza. Las técnicas de *alineamiento textual* facilitan la detección de esta redundancia a partir de una transformación

del bitexto original en una representación equivalente que relaciona las palabras que representan traducciones mutuas en cada uno de los idiomas.

En el capítulo actual se han presentado sendas propuestas de compresión basadas en el modelado de los bitextos alineados como base para la detección y eliminación de esta redundancia semántica. Tanto TRC como 2LCAB son técnicas basadas en la experiencia derivada de la compresión de texto. Mientras que TRC mantiene un modelado orientado a palabras, 2LCAB define el concepto de *bipalabra* (Definición 6.1) como un símbolo de representación orientado al significado compartido (concepto) por las dos palabras que la constituyen.

Las técnicas TRC interpretan la secuencia de bipalabras, que define el bitexto, como las dos secuencias de palabras que comprenden cada uno de los textos del bitexto. El texto origen ($\mathcal{L}$) se modela y codifica por sí mismo atendiendo a sus propiedades como lenguaje natural. Para este propósito se consideran dos propuestas, orientadas a palabras, de orden 0 y orden 1 respectivamente. La primera de ellas aproxima un Huffman de palabras [TM97], mientras que la segunda adapta la técnica E-G$_1$ (presentada en el capítulo anterior) de tal forma que la utilización de la palabra ϵ no deteriore la información representada en su grafo de palabras. Por su parte, el texto meta ($\mathcal{R}$) se codifica utilizando la relación de traducción subyacente al bitexto, c de tal forma que cada palabra en $\mathcal{R}$ se codifica de acuerdo al contexto representado por su traducción en $\mathcal{L}$. La combinación de estas propuestas dan como resultado dos técnicas de compresión referidas como TRC$_{(0,1)}$ y TRC$_{(1,1)}$. La primera de ellas muestra una mayor velocidad de compresión y descompresión aprovechando el modelado de orden 0 utilizado para $\mathcal{L}$ y obtiene tasas de compresión del $[14\,\%, 17\,\%]$ para pares de idiomas próximos y del $[18\,\%, 20,5\,\%]$ para los restantes. Por su parte, TRC$_{(1,1)}$ mejora la efectividad anterior a costa de aumentar los tiempos de compresión y descompresión dada la necesidad de construir y manejar el grafo Edge-Guided utilizado para $\mathcal{L}$. Sus rangos de compresión se desplazan hasta el $[10,5\,\%, 15\,\%]$ para idiomas próximos y del $[16\,\%, 19\,\%]$ para el resto. Los resultados obtenidos por ambas técnicas permiten concluir su utilidad ante las diferentes políticas de alineamiento consideradas dado que son capaces de manejar vocabularios de gran tamaño sin que eso suponga un impacto importante en sus tasas de compresión.

Los modelos basados en bipalabras surgen como respuesta a la necesidad de utilizar la información contenida en el bitexto con independencia del idioma en el que esté representada. Las técnicas anteriores representan el bitexto supeditando $\mathcal{R}$ a su relación de traducción en $\mathcal{L}$. Esta decisión dificulta un eventual acceso al bitexto a través de $\mathcal{R}$ aunque mantiene la posibilidad de hacerlo a través de $\mathcal{L}$. Los modelos de bipalabras representan el bitexto a través de sus relaciones de traducción mutua de tal forma que el planteamiento de vocabulario en dos niveles, propuesto en 2LCAB, facilita el acceso al contenido con independencia del idioma suministrado en la consulta. La efectividad conseguida por 2LCAB$_{huff}$ es comparable a la obtenida por TRC$_{(0,1)}$ utilizando técnicas de alineamiento *uno a uno*. Sus rangos de compresión se distribuyen en $[13,5\,\%, 17\,\%]$ para idiomas próximos y en $[18,5\,\%, 20\,\%]$ para los restantes. Sin embargo la consideración de multipalabras en el alineamiento del bitexto causa un crecimiento elevado del tamaño del vocabulario de bipalabras, deteriorando las tasas de compresión que obtiene 2LCAB$_{huff}$. Mientras que para idiomas próximos se mantiene el rango $[13,5\,\%, 17\,\%]$ (dado que apenas se requiere la utilización multipalabras en el alineamiento del bitexto), para el resto de idiomas se pasa a un $[19\,\%, 22,5\,\%]$. En lo que respecta a su eficiencia, 2LCAB$_{huff}$ es la opción más rápida.

La utilización de una codificación orientada a byte (ETDC) sobre las propiedades de representación de 2LCAB supone la obtención de la técnica 2LCAB$_{etdc}$ que posibilita operar sobre el bitexto comprimido. Desde la perspectiva espacial, sustituir la codificación de Huffman orientada a bit (utilizada en 2LCAB$_{huff}$) por ETDC aumenta entre 2 y 3 puntos porcentuales el tamaño final del bitexto comprimido a cambio de ganar un acceso eficiente a la información. 2LCAB permite ejecutar cuatro operaciones diferentes sobre el bitexto comprimido:

- Identificación de todas las traducciones de una palabra en el idioma complementario.
- Localización de todas las ocurrencias de una palabra en el bitexto con independencia de su significado.
- Localización de todas las ocurrencias de una palabra en el bitexto cuyo significado coincida con la información etiquetada por su traducción en el idioma complementario.
- Extracción de *snippets* en el mismo idioma en el que se lleva a cabo la consulta y/o en su complementario.

Este conjunto de operaciones permiten considerar la utilización de `2LCAB` como motor de búsqueda en aplicaciones de búsqueda de concordancias bilingües como la descrita en [Bar04]. Este tipo de herramientas se utilizan directamente por usuarios humanos como complementos de ayuda a la traducción en diferentes contextos de aplicación. Esto posibilita poder operar con alineamientos sencillos *uno a uno*, como los actualmente manejados por `2LCAB`, dado que la información que se aporta a través de los *snippets* le otorga al usuario el contexto necesario para cubrir las necesidades planteadas en este tipo de aplicaciones.

Las políticas de alineamiento *unoa a muchos*, analizadas en el capítulo actual, permiten obtener representaciones de los bitextos más orientadas a su utilización en aplicaciones de carácter automático. La experimentación llevada a cabo sobre las técnicas `TRC` y `2LCAB` han demostrado que la política `1:N Cont.` obtiene un comportamiento satisfactorio incluso con las técnicas actuales. Sin embargo, `1:N No Cont.` requiere tamaños de diccionario muy elevados lo que aumenta la complejidad de la representación y dificulta su codificación con las técnicas actuales cuya efectividad se reduce, sobre todo para `2LCAB`.

Aún así, la experiencia adquirida con la representación de este tipo de alineamientos en las técnicas actuales sugiere ideas de futuro muy interesantes. Los modelos de traducción automática se caracterizan por unos altos costes espaciales lo cual dificulta tanto su manejo como su explotación en contextos de aplicación *on-line*. Aunque existen algunos trabajos que intentan reducir estos órdenes de complejidad mediante estructuras de datos más avanzadas [Lop08], su éxito parece limitado ante aplicaciones como la traducción de consultas requeridas en contextos translingües de recuperación de información. Esto conlleva la necesidad de estudiar y diseñar un modelo de representación para los alineamientos de tipo *muchos a muchos* que permita obtener tanto un almacenamiento compacto como un acceso eficiente a la información contenida en el bitexto de forma que ésta puede ser utilizada en contextos como el anterior y aquellos otros que precisen de mecanismos eficientes de traducción automática como, por ejemplo, los relacionados con *memorias de traducción* en los que se suele operar a nivel de segmentos como frases completas o párrafos.

Volveré a ese lugar
donde nací.

Antonio Vega

7

Conclusiones y Trabajo Futuro

La gran cantidad de información textual manejada en entornos globales como la WWW, u otros más específicos como las bibliotecas digitales o las bases de datos documentales, precisa de un almacenamiento compacto y un procesamiento eficiente en su ámbito de aplicación. La compresión de texto se plantea como la solución más adecuada ante esta problemática considerando tanto su capacidad para la reducción de espacio como el ahorro temporal obtenido al manejar representaciones comprimidas.

El presente trabajo afronta este contexto a partir de la definición de diferentes técnicas de compresión de texto orientadas a palabra cuya aportación se puede evaluar desde dos perspectivas complementarias: la compresión genérica de textos en lenguaje natural y la especialización llevada a cabo en la compresión de corpus paralelos bilingües.

La utilización de estadísticas de orden superior, como base para el modelado y codificación de un texto permite, obtener técnicas capaces de obtener unas mejores tasas de compresión. Sin embargo, el modelado de orden superior orientado a palabras puede convertirse en un problema impracticable en términos de tiempo de cómputo y espacio en memoria debido al numeroso conjunto de relaciones existentes entres las palabras que conforman un texto. Por lo tanto, afrontar este problema requiere de la utilización de heurísticas que permitan identificar las relaciones más importantes y conseguir, a través de ellas, un modelado de orden superior cuyo *trade-off* espacio/tiempo sea competitivo con técnicas ya existentes.

- La primera de las propuestas realizadas en esta tesis: *Word-Codeword Improved Mapping* (`WCIM`), se basa en la utilización de cuatro heurísticas específicas para la gestión de diccionarios de gran tamaño cuya distribución de símbolos sea de naturaleza *power-law*. Los textos en lenguaje natural encajan en este perfil dado el gran número de palabras diferentes identificadas en un texto (cuya distribución se aproxima a través de la Heaps [Hea78, BYRN99]) y la naturaleza asimétrica que sigue su distribución de probabilidad (estudiada a través de la Ley de Zipf [Zip49, BYRN99]). `WCIM` no es una técnica de compresión por sí misma sino una forma de preprocesar el texto que permite transformarlo en una representación equivalente más redundante que la original. La compresión de este resultado, con una técnica orientada a bit (como `bzip2`, `ppmdi` o `p7zip`), es la que permite alcanzar las tasas de compresión (inferiores al 19 %) reportadas en el capítulo 4. Resulta interesante observar como el estudio experimental llevado a cabo demuestra la alta efectividad que muestra `p7zip` para la compresión de estas representaciones, superando los resultados alcanzados al utilizar técnicas predictivas como `ppmdi`. La experimentación desarrollada con `WCIM` ha permitido identificar otras posibilidades de compresión para `mPPM`, de tal forma que esta experiencia se puede utilizar como base para el desarrollo de una nueva técnica, de carácter semi-estático, capaz de integrar los puntos fuertes tanto de `mPPM` como de `WCIM` para su compresión final con `p7zip`. De forma complementaria, la utilización de un conjunto limitado de q-gramas para la extensión del alfabeto original plantea una propuesta capaz de obtener, a priori, unas ratios de compresión mejores a las obtenidas actualmente.

Las heurísticas integradas en WCIM adaptan propiedades del texto que pueden ser consideradas en otros contextos de aplicación con características similares. La gestión de símbolos poco frecuentes en técnicas basadas en diccionario permite reducir notablemente el tamaño de dicha estructura y con ello reducir la longitud media de palabra de código utilizada para la codificación del mensaje de entrada. Actualmente esta heurística ha sido integrada satisfactoriamente en sendos modelos de compresión: 1) en la técnica TRC se utiliza para descartar las traducciones que se utilizan una única vez en el vocabulario de traducción de una determinada palabra; 2) en un prototipo centrado en la compresión de RDF[1] que la utiliza para la gestión de los símbolos (literales, URIs o "blancos") que aparecen en un único triple. Esta decisión permite obtener un diccionario principal un 40 % menor que el utilizado originalmente, reduciendo de forma notable la longitud media de palabra de código utilizada para la representación del grafo RDF con la nueva configuración del diccionario.

- Las técnicas Edge-Guided (E-G) complementan la propuesta WCIM, diseñando un modelo específico para la representación de las estadísticas de orden superior identificadas en un texto. El grafo E-G representa el texto a partir de las relaciones de adyacencia existentes entre las palabras que lo conforman y las codifica utilizando un vocabulario de transiciones independiente para cada nodo del modelo. De esta manera se obtiene un modelo de orden 1 (E-G$_1$) cuyo *trade-off* espacio/tiempo ya es capaz de mejorar el obtenido por técnicas de orden superior orientadas a caracteres como ppmdi cuya tasas de compresión (para textos de mediano-gran tamaño) son entre 3 y 4 puntos porcentuales peores que la técnica propuesta. Tanto el modelo de grafo propuesto por E-G$_1$ como el esquema de codificación considerado se utilizan como base para el desarrollo de una propuesta de orden superior.

E-G$_k$ desarrolla la idea de construir un modelo de orden 1 orientado a frases. Sin embargo E-G$_k$ no construye un modelo completo de frases sino que únicamente considera las más significativas. Su representación considera la utilización de una gramática libre de contexto obtenida a partir de una variante de algoritmo Re-Pair [Lar99, LM00]. La jerarquía de palabras que describe cada frase se integra en el grafo mediante la descomposición de la gramática en "heavy paths" [HT84], de forma que estas palabras se pueden utilizar en un mecanismo de *blending* que guarda ciertas similitudes con el utilizado en PPM. Dicho mecanismo permite descender por la jerarquía con el fin de representar un símbolo que no ha sido localizado en los contextos superiores. Las tasas de compresión obtenidas por este técnica se reducen hasta valores del 18 %, a costa de unos procesos de compresión cuya eficiencia se sitúa entre la obtenida por ppmdi y p7zip, mejorando la alcanzada por el algoritmo Re-Pair original. Los procesos de descompresión tienen una velocidad de cómputo siendo comparable con técnicas como WCIM. A tenor de los resultados obtenidos, se concluye que las frases dejan de ser significativas a partir de un límite de $4-5$ palabras de forma que añadir estos símbolos al alfabeto no mejora sustancialmente la efectividad de la técnica y en el algunos casos incluso la deteriora.

El trabajo futuro relativo a las técnicas E-G radica en la utilización del modelo del lenguaje obtenido en E-G$_k$. La información representada en el grafo se puede utilizar en aplicaciones relacionadas con la extracción de palabras clave y la catalogación de contenidos [MPAdlF08]. La idea principal se centra en integrar en dicho modelo técnicas como TextRank [MS05], centradas en la evaluación de la importancia de cada arista en el grafo y su posterior utilización para la identificación de estas palabras claves con las que realimentar una aplicación semi-automática para la catalogación de contenidos.

- Finalmente, la especialización de lenguaje natural que muestran los textos bilingües aportan

[1] http://www.w3.org/RDF/

un conjunto de resultados y conclusiones de interés para el avance del trabajo en este campo. Como se indicaba al principio del capítulo anterior, existe poco trabajo previo centrado en la representación compacta de bitextos, por lo que las aportaciones presentadas en esta tesis complementa la experiencia anterior con sendas propuestas específicas orientadas a palabras y bipalabras. Nótese la importancia que tiene, en estas técnicas, la integración del conocimiento derivado de utilizar políticas de alineamiento textual sobre el bitexto original.

La técnica TRC (*Translation Relationship-based Compressor*) considera la experiencia previa presentada por Nevill-Manning y Bell [NMB92] para definir un modelo orientado a palabras capaz de representar la relación de traducción aportada en el bitexto alineado. TRC plantea dos alternativas para la compresión del texto origen de acuerdo a sus propiedades como lenguaje natural: un código Huffman orientado a palabras ($\texttt{TRC}_{(0,1)}$) y una adaptación de la técnica $\texttt{E-G}_1$ ($\texttt{TRC}_{(1,1)}$). En ambos casos, el texto meta $\mathcal{R}$ se comprime utilizando el texto origen $\mathcal{L}$ como contexto de representación, obteniendo un modelado de orden 1 sobre la relación de traducción entre las palabras del bitexto. $\texttt{TRC}_{(1,1)}$ mejora entre 2 y 4 puntos porcentuales la efectividad alcanzada por $\texttt{TRC}_{(0,1)}$ a costa de aumentar, ligeramente, sus tiempos de compresión y descompresión. Ambas técnicas se caracterizan por un *trade-off* espacio/tiempo que mejora el alcanzado por las técnicas universales consideradas. Por su parte, *2-Level Compressor for Aligned Bitexts* (2LCAB) considera el concepto de *bipalabra* (Definición 6.1) como símbolo de representación orientado a la relación de traducción inherente al bitexto. Cada bipalabra contiene dos palabras que son traducción mutua en el bitexto. Por lo tanto, lo modelos de bipalabras enfocan la representación de la semántica del bitexto. Esta política de modelado permite aislar la información contenida en el bitexto del idioma en el que ésta se expresa. Esta decisión permite un acceso a la información independiente del idioma utilizado. La efectividad de 2LCAB sobre una codificación de Huffman es comparable a la obtenida por $\texttt{TRC}_{(0,1)}$. Sin embargo, estas tasas de compresión se deterioran en 2LCAB al considerar la utilización de multipalabras en $\mathcal{R}$.

El modelado 2LCAB se combina, a su vez, con una codificación orientada a byte (ETDC) aumentando en $2-3$ puntos porcentuales las tasas de compresión anteriores. 2LCAB plantea la primera técnica de compresión cuya representación del bitexto puede ser accedida en formato comprimido. Esta técnica soporta consultas eficientes que permiten recuperar todas las traducciones de una palabra en el idioma complementario, las ocurrencias en el bitexto de la palabra solicitada (así como la de sus traducciones) y, de entre ellas, seleccionar sólo aquellas que poseen un determinado significado. Complementariamente, 2LCAB permite la extracción de *snippets* con los que contextualizar los resultados de cada una de las operaciones anteriores de búsqueda. Este conjunto de operaciones facilitan la utilización actual de 2LCAB como motor de una herramienta de búsqueda de concordancias bilingües enfocada a su utilización en contextos de apoyo a la traducción.

Finalmente, la idea principal de trabajo futuro para el campo de los textos bilingües se centra en la extensión de los modelos actuales de tal forma que soporten la representación eficiente de alineamientos $N:M$. Esto es, alineamientos que establecen la relación de traducción entre N palabras en el texto origen y M en el texto meta. Este tipo de representación es la base para el desarrollo de herramientas automáticas de traducción cuya aplicación puede llevarse a cabo en diferentes ámbitos como, por ejemplo, entornos de recuperación translingüe o aplicaciones como las memorias de traducción cuya operativa se centra principalmente en la manipulación de segmentos como frases completas o párrafos.

7.1 Publicaciones relacionadas

A continuación se plantea una revisión las publicaciones llevadas a cabo sobre el trabajo presentado en la presente tesis. Dichas publicaciones se clasifican para cada una de las propuestas

presentadas en los capítulos 4, 5 y 6 del documento actual. La clasificación de los congresos referidos se ajusta a lo establecido en el *Ranking Conference Core*[2].

Capítulo 4: los resultados preliminares de la técnica $\texttt{WCIM}_{ppmdi}$ han sido publicados como artículo en un congreso internacional de nivel `A+`:

- *Joaquín Adiego, Miguel A. Martínez-Prieto, Pablo de la Fuente. High Performance Word-Codeword Mapping Algorithm on PPM. In* Data Compression Conference (DCC 2009)*, páginas 23–32, 2009.*

Capítulo 5: una revisión completa de la familia de compresores $\texttt{E-G}_1\texttt{+X}$ y sus resultados sobre diferentes técnicas universales se encuentra actualmente en revisión en la revista *Information Sciences*. Además, un resultado previo basado en $\texttt{E-G}_1\texttt{+ppmdi}$ ha sido publicado como artículo en un congreso internacional clasificado como `B`.

- *Miguel A. Martínez-Prieto, Joaquín Adiego, Pablo de la Fuente. On Compression of Natural Language Text with Edge-Guided Methods.* Information Sciences*, Elsevier (en periodo de revisión).*

- *Joaquín Adiego, Miguel A. Martínez-Prieto, Pablo de la Fuente. Edge-Guided Natural Language Text Compression. In* 14th String Processing and Information Retrieval Symposium (SPIRE 2007)*, LNCS, páginas 14–25, 2007.*

Asimismo, parte de los resultados presentados, en este trabajo de tesis, para la técnica $\texttt{E-G}_k$ han sido aceptados para su presentación como póster en un congreso internacional clasificado como `A+` y publicados como artículo en un congreso de carácter nacional:

- *Miguel A. Martínez-Prieto, Joaquín Adiego, Pablo de la Fuente, Javier D. Fernández. High-Order Text Compression on Hierarchical Edge-Guided. In* Data Compression Conference (DCC 2010)*, página 543, 2010.*

- *Miguel A. Martínez-Prieto, Joaquín Adiego, Pablo de la Fuente. Un modelo para el análisis y explotación de información cognitiva en repositorios documentales. In* XIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2008)*, páginas 75–86, 2008.*

Capítulo 6: una versión preliminar de la técnica `2LCAB`, junto con algunas de sus posibilidades de búsqueda, ha sido publicada como artículo en un congreso internacional clasificado como `B`. Por otra parte, los resultados previos sobre los que se asienta el uso de estas políticas de alineamiento como paso previo a la compresión de un bitexto han sido publicados como sendos pósters en un congreso internacional `A+`.

- *Joaquín Adiego, Nieves R. Brisaboa, Miguel A. Martínez-Prieto, Felipe Sánchez-Martínez. A two-level structure for compressing aligned bitexts. In* 16th International Symposium on String Processing and Information Retrieval (SPIRE 2009)*, páginas 114–121, 2009.*

- *Joaquín Adiego, Miguel A. Martínez-Prieto, Javier E. Hoyos-Torío, Felipe Sánchez-Martínez. Modelling Parallel Texts for Boosting Compression. In* Data Compression Conference (DCC 2010)*, página 517, 2010.*

- *Miguel A. Martínez-Prieto, Joaquín Adiego, Felipe Sánchez-Martínez, Pablo de la Fuente, Rafael C. Carrasco. On the use of word alignments to enhance bitext compression. In* Data Compression Conference (DCC 2009)*, página 459, 2009.*

[2] http://www.cs.wm.edu/∼srgian/tier-conf-final2007.html

Aunque tú no lo entiendas
nunca escribo el remite
en el sobre.

Quique González

A Entorno Experimental

A.1 Máquina de experimentación y propiedades de compilación

Todos los resultados mostrados en el presente trabajo de tesis han sido obtenidos bajo el mismo entorno de experimentación. La máquina considerada para el desarrollo de estas pruebas utiliza un procesador Intel Core2Duo P8600@2,40GHz, 3MB de memoria cache y 4GB de memoria DDR2-800 RAM. El sistema operativo es `Ubuntu 9.04` - *Jaunty Jackalope*[1] (kernel 2.6.28-17 generic) con arquitectura `x86_64-linux-gnu`.

Todos los prototipos desarrollados están escritos en lenguaje `C++` y compilados con `g++ versión 4.3.3.`, optimización `-O9` y opción `-m32`.

A.2 Textos utilizados

A.2.1. Colecciones de Lenguaje Natural

Las colecciones de lenguaje natural consideradas para los experimentos mostrados en los capítulos 4 y 5 se incluyen en el conjunto de textos de prueba publicado por la iniciativa `TREC` (*Text REtrieval Conference*)[2].

A continuación se detallan las diferentes colecciones utilizadas. Nótese que para `AP`, `WSJ` y `ZIFF` se consideran textos de tamaño aproximado 1, 5, 10, 20, 40, 60 y 100 megabytes (MBytes) obtenidos a partir de la colección completa. Esto supone que el fichero `WSJ100` está formado por los 100 primeros MBytes de la colección `WSJ` y así para el resto de colecciones y tamaños. Para cada una de las colecciones se introduce su origen y se muestra una descripción de su contenido a partir de las estadísticas que representan su tamaño en MBytes, el número de palabras diferentes que definen el vocabulario de cada texto (`Voc.`) y el número *total* de palabras identificadas (atendiendo a lo caracterizado en la definición 3.1, página 34, y considerando que todas nuestras propuestas representan el texto sobre la transformación *spaceless words*).

Associated Press (`AP`). Contiene documentos de *Associated Press* correspondientes al año 1988. Está incluida en la colección `TREC-2`. Su descripción se muestra en la tabla A.1.

Wall Street Journal (`WSJ`). Contiene documentos de *Wall Street Journal* correspondientes a los años 1990, 1991 y 1992. Está incluida en la colección `TREC-2`. Su descripción se muestra en la tabla A.1.

[1] http://www.ubuntu.com/
[2] http://trec.nist.gov/

Texto	MBytes	Palabras		Texto	MBytes	Palabras		Texto	MBytes	Palabras	
		Voc.	Total			Voc.	Total			Voc.	Total
AP001	1, 13	19352	252066	WSJ001	1, 17	18858	257794	ZIFF001	0, 97	13153	222906
AP005	5, 54	41684	1230967	WSJ005	5, 26	39873	1162516	ZIFF005	5, 80	36157	1334998
AP010	9, 98	54542	2217303	WSJ010	10, 02	53832	2217389	ZIFF010	10, 65	51891	2444512
AP020	20, 24	74342	4497423	WSJ020	20, 25	74008	4475571	ZIFF020	20, 32	72805	4671618
AP040	40, 55	101957	8997915	WSJ040	40, 69	100784	8988554	ZIFF040	40, 68	105233	9335155
AP060	60, 41	122639	13406790	WSJ060	59, 52	119454	13166345	ZIFF060	60, 05	128539	13784287
AP100	100, 15	157381	22213576	WSJ100	100, 08	149658	22161021	ZIFF100	100, 81	169234	23181076

Tabla A.1: Descripción de las colecciones AP, WSJ y ZIFF.

Texto	MBytes	Palabras	
		Voc.	Total
CR	48, 72	117718	10113132
FT91	14, 07	75681	3059619
FT92	167, 32	284061	36520173
FT93	188, 43	291429	41771937
FT94	194, 34	295018	43039675

Tabla A.2: Descripción de las colecciones CR y FT.

Computer Select Disks / Ziff-Davis (ZIFF). Contiene documentos de *Computer Select Disks* (copyrighthed Ziff-Davis) correspondientes a los años 1989 y 1990. Está incluida en la colección TREC-2. Su descripción se muestra en la tabla A.1.

Congressional Record (CR). Contiene documentos de *Congressional Record* correspondientes al 103^{rd} *Congress* del año 1993. Está incluida en la colección TREC-4. Su descripción se muestra en la tabla A.2.

Financial Times Limited (FT). Contiene documentos de *Financial Times Limited* correspondientes a los años 1991 (FT91), 1992 (FT92), 1993 (FT93) y 1994 (FT94). Está incluida en la colección TREC-4. Su descripción se muestra en la tabla A.2.

A.2.2. Corpus Paralelos Bilingües

Las colecciones actuales de bitextos se eligen de acuerdo a los requisitos establecidos en la experimentación llevada a cabo en la Sección §6.5. Esto supone la necesidad de disponer de un conjunto variado de pares de idiomas que posibilite el análisis de las relaciones existentes entre cada uno de ellos y el efecto que éstas tienen en las técnicas planteadas para su compresión.

El grueso de los pares de idiomas considerados proceden de la colección *Europarl* (*European Parliament Proceedings Parallel Corpus*)[3][Koe05] extraída a partir de las actas del Parlamento Europeo. Se considera la utilización de seis pares de estos idiomas: alemán-inglés (de-en), francés-inglés (fr-en), inglés-castellano (en-es), castellano-francés (es-fr), castellano-italiano (es-it) y castellano-portugués (es-pt).

A continuación se muestra un resumen de sus propiedades estadísticas. Para cada uno de los pares de idiomas se consideran bitextos de tamaño aproximado 1, 10 y 100 MBytes y se indica el número de palabras (incluyendo ϵ) que componen el vocabulario de sus textos *origen* y *meta*, así como el número total de bipalabras obtenidas para su representación de acuerdo a las cuatro políticas propuestas a nivel experimental (ver §6.5).

Finalmente se consideran dos conjuntos de bitextos adicionales en representación de pares de idiomas muy próximos entre sí. Se utilizan sendos bitextos *castellano-catalán* (es-ca) y

[3]http://www.statmt.org/europarl/

Bitexto	MBytes	Palabras (Voc.)		Bipalabras (Total)			
		Origen	Meta	1:1		1:N	
				No Desp.	Desp.	Cont.	No Cont.
de-en	0,98	9961	6515	128924	119150	103580	95458
	9,93	39291	19306	1301305	1203647	1050854	971987
	100,82	139010	51015	13120052	12106577	10602529	9852297
en-es	0,98	6697	9170	131240	122046	109280	104878
	9,94	19466	30487	1330748	1238496	1106532	1057681
	101,37	51352	81868	13214990	12298572	11072964	10676325
es-fr	0,99	8717	8031	122653	117695	105215	99655
	9,94	29550	25416	1233229	1183235	1062594	1008361
	100,93	80416	65225	12487356	11979311	10747408	10189344
es-it	0,99	8801	8750	121341	116764	106092	101179
	9,91	29759	28467	1218160	1173149	1067992	1017694
	99,35	80431	74633	12086365	11652137	10642364	10171300
es-pt	0,97	8667	8725	115715	111778	102923	98907
	9,82	29363	29058	1172633	1130687	1041572	999465
	99,19	80173	80195	11751300	11336391	10468679	10060951
fr-en	0,98	8212	6494	132434	122585	111689	107599
	9,79	25537	19046	1326975	1229074	1122212	1082218
	100,04	65875	50411	13378814	12394893	11344378	10955977

Tabla A.3: Descripción de los bitextos procedentes de la colección *Europarl*.

Bitexto	MBytes	Palabras (Voc.)		Bipalabras (Total)			
		Origen	Meta	1:1		1:N	
				No Desp.	Desp.	Cont.	No Cont.
es-ca	1,12	15595	14940	125734	125328	119880	119178
	10,98	54116	52257	1228109	1223765	1171656	1165515
	100,01	161127	159217	11351053	11312923	10843650	10783467

Tabla A.4: Descripción de los bitextos procedentes de *El Periódico de Catalunya*.

Bitexto	MBytes	Palabras (Voc.)		Bipalabras (Total)			
		Origen	Meta	1:1		1:N	
				No Desp.	Desp.	Cont.	No Cont.
es-gl	1,04	6488	6543	110188	110183	109667	109648
	10,06	24981	25284	1069537	1069390	1063950	1063459

Tabla A.5: Descripción de los bitextos procedentes del *Diario Oficial de Galicia*.

castellano-gallego (`es-gl`) procedentes, respectivamente, de *El Periódico de Catalunya*[4] (publicación diaria en castellano y catalán; los textos utilizados corresponden a la colección del año 2000) y el *Diario Oficial de Galicia*[5] (los textos considerados proceden de las órdenes publicadas entre Mayo y Octubre de 2006; nótese que, a diferencia de los anteriores, el bitexto de mayor tamaño considerado para el caso actual es de 10 MBytes). Las tablas A.4 y A.5 describen las propiedades de estos bitextos de acuerdo a las mismas estadísticas consideradas anteriormente.

A.3 Herramientas Comparativas

La sección actual plantea un breve resumen de los diferentes compresores utilizados en los diferentes procesos de experimentación expuestos en los capítulos 4, 5 y 6.

A.3.1. bzip2

`bzip2`[6] es un compresor *open-source* desarrollado por Julian Seward. La primera versión de `bzip2` (versión `0.15`) se publicó en Julio de 1996, mientras que la utilizada en la experimentación actual

[4] http://www.elperiodico.com/
[5] http://www.xunta.es/diario-oficial
[6] http://www.bzip.org/

es la versión `1.0.5` publicada en Diciembre de 2007.

`bzip2` procesa el conjunto de datos de entrada mediante bloques de tamaño comprendido entre 100 y 900 KBytes. Cada uno de estos bloques se utiliza como entrada a un proceso que ejecuta la transformada de *Burrows-Wheeler* obteniendo una representación equivalente del mensaje sobre la que se posteriormente se aplica la transformación *move-to-front* y, finalmente, una codificación de `Huffman`. El rendimiento de `bzip2` es asimétrico. El proceso de compresión está está fuertemente gravado por la ejecución de la transformada de *Burrows-Wheeler* directa mientras que el descompresor aprovecha la mayor velocidad de la transformada inversa.

En términos generales, `bzip2` se muestra como una alternativa más efectiva que `gzip` o `lzw`. Sus tasas de compresión son comparables a la obtenidas por el algorimo `lzma` implementado en `p7zip` e inferiores a las obtenidas por los algoritmos predictivos de la familia `PPM` como `ppmdi`.

A.3.2. gzip

`gzip`[7] es un compresor desarrollado, en sus inicios, por Jean-Loup Gailly and Mark Adler y cuya primera versión se remonta a Octubre de 1992. La versión utilizada en la experimentación actual es la `1.3.12-6 ubuntu2`.

`gzip` está basado en el algoritmo `deflate` que combina un modelado de diccionario `LZ77` y codificación de `Huffman`. Este algoritmo se diseñó como alternativa a `LZW` y otros algoritmos patentados y, por tanto, con una capacidad de uso limitada. `gzip` representa la alternativa más eficiente de las consideradas para la experimentación llevada a cabo en este trabajo. Sin embargo, sus tasas de compresión no son competitivas con las obtenidas por `bzip2`, `p7zip` y `ppmdi`. Aún así, se ha considerado su utilización dado que plantea un compresor de referencia en el contexto de la compresión de datos.

A.3.3. p7zip

`p7zip`[8] define un entorno completo de herramientas relacionadas con la compresión de datos y otras utilidades complementarias. Desarrollado por Igor Pavlov, la principal aportación de este software es el algoritmo `LZMA`. La experimentación actual se ha llevado a cabo con la versión `4.58d̃fsg1.1`.

El algoritmo `LZMA` plantea una variante del algoritmo `LZ77` que elimina el concepto de ventana deslizante a costa de utilizar un diccionario cuyo tamaño se limita a 1GB pero podría extenderse hasta los 4GB. Complementariamente, `LZMA` caracteriza de forma específica la codificación de los triples, alcanzando una efectividad notablemente superior a la obtenida por otros compresores de la familia `LZ`. Los procesos de compresión `LZMA` requieren tiempos de cómputo muy elevados mientras que la descompresión muestra un rendimiento extraordinariamente eficiente respecto a la compresión alcanzado una eficiencia comparable a la obtenida por `gzip`.

Su efectividad tiende a ser inferior a la obtenida por `ppmdi` aunque en algunos casos llega a superarla. En términos generales obtienen unas ratios de compresión comparables a las de `bzip2`.

A.3.4. ppmdi

`ppmdi` es un compresor obtenido a partir de la implementación del método `D` para la gestión de códigos de escape en `PPM`. La implementación utilizada para la experimentación actual fue

[7] http://www.gzip.org/
[8] http://www.7-zip.org/

desarrollada por Dick Cheney para la implementación de `XMLPPM` y está públicamente disponible en el sitio de y *Pizza&Chili*[9].

`ppmdi` tiende a obtener las mejores tasas de compresión entre las técnicas consideradas. Se caracteriza por una menor eficiencia que el resto de técnicas consideradas de forma que sus tiempos de compresión sólo son más bajos que los requeridos por `p7zip` mientras que en descompresión representan siempre la alternativa menos eficiente de las cuatro consideradas.

[9] http://pizzachili.dcc.uchile.cl/experiments.html

Bibliografía

[ABMPSM09] J. Adiego, N. R. Brisaboa, M. A. Martínez-Prieto, and F. Sánchez-Martínez. A two-level structure for compressing aligned bitexts. In *16th International Symposium on String Processing and Information Retrieval (SPIRE 2009)*, pages 114–121, 2009.

[Abr63] N. Abramson. *Information Theory and Coding*. McGraw-Hill, 1963.

[ÁCPFL09] S Álvarez, A. Cerdeira-Pena, A. Fariña, and S. Ladra. Desarrollo de un compresor de textos orientado a palabras basado en ppm. In *XIV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2009)*, pages 237–248, 2009.

[Adi05] J. Adiego. *La estructura de los documentos en el ámbito de la recuperación de información: propuestas para su compresión, indexación y recuperación*. PhD thesis, Departamento de Informática, Universidad de Valladolid, España, 2005.

[AdlF06] J. Adiego and P. de la Fuente. Mapping Words into Codewords on PPM. In *13th String Processing and Information Retrieval Symposium (SPIRE 2006)*, LNCS, pages 181–192, 2006.

[AL00] A. Apostolico and S. Lonardi. Off-line Compression by Greedy Textual Substitution. *Proceedings of the IEEE*, 88(11):1733–1744, 2000.

[AMPdlF07] J. Adiego, M.A. Martínez-Prieto, and P. de la Fuente. Edge-Guided Natural Language Text Compression. In *14th String Processing and Information Retrieval Symposium (SPIRE 2007)*, LNCS, pages 14–25, 2007.

[AMPdlF09] J. Adiego, M.A. Martínez-Prieto, and P. de la Fuente. High Performance Word-Codeword Mapping Algorithm on PPM. In *Data Compression Conference (DCC 2009)*, pages 23–32, 2009.

[AT05] J. Abel and W.J. Teahan. Universal text preprocessing for data compression. *IEEE Transactions on Computers*, 54:497–507, 2005.

[AZM+01] F. S. Awan, N. Zhang, N. Motgi, R. T. Iqbal, and A. Mukherjee. LIPT: A Reversible Lossless Text Transform to Improve Compression Performance. In *Data Compression Conference (DCC 2001)*, 2001.

[Bar04] G.M. Barlow. Parallel Concordancing and Translation. In *Proc. 26th International Conference on Translating and the Computer*, 2004.

[BCW90] T. Bell, J. Cleary, and I. Witten. *Text Compression*. Prentice Hall, 1990.

[BFL+10] N. Brisaboa, A. Fariña, J.R. López, G. Navarro, and E. R.López. A new searchable variable-to-variable compressor. In *Data Compression Conference (DCC 2010)*, pages 199–208, 2010.

[BFNE03] N.R. Brisaboa, A. Fariña, G. Navarro, and M. Esteller. (S,C)-Dense Coding: An Optimized Compression Code for Natural Language Text Databases. In *Proc. 10th International Symposium on String Processing and Information Retrieval (SPIRE 2003)*, pages 122–136, 2003.

[BFNP05] N.R. Brisaboa, A. Fariña, G. Navarro, and J.L. Paramá. Efficiently decodable and searchable natural language adaptive compression. In *28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2005)*, pages 234–241, 2005.

[BFNP07] N.R. Brisaboa, A. Fariña, G. Navarro, and J.R. Paramá. Lightweight natural language text compression. *Information Retrieval*, 10(1):1–33, 2007.

[BFNP08] N.R. Brisaboa, A. Fariña, G. Navarro, and J. Paramá. New adaptive compressors for natural language text. *Software Practice and Experience*, 38(13):1429–1450, 2008.

[BFNP10] N.R. Brisaboa, A. Fariña, G. Navarro, and J. Paramá. Dynamic lightweight text compression. *ACM Transactions on Information Systems*, 28(3):artículo 1, 2010.

[BINP03] N. Brisaboa, E. Iglesias, G. Navarro, and J. Paramá. An efficient compression code for text databases. In *25th European Conference on Information Retrieval Research (ECIR 2003)*, pages 468–481, 2003.

[BM77] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Communications of ACM*, 20(10):762–772, 1977.

[Bob06] V. Bobicev. Text Classification Using Word-Based PPM Models. *The Computer Science Journal of Moldova*, 14(2):183–201, 2006.

[BPPM93] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

[BSTW86] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.

[BW94] M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

[BW99a] J. Bach and I.H. Witten. Lexical attraction for text compression. (Working paper 99/01). Hamilton, New Zealand: University of Waikato, Department of Computer Science., 1999.

[BW99b] J. Bach and I.H. Witten. Lexical attraction for text compression. In *Data Compression Conference*, page 516, 1999.

[BWC89] T.C. Bell, I.H. Witten, and J.G. Cleary. Modeling for text compression. *ACM Computing Surveys*, 4(21):557–591, 1989.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[Car03] J. Carroll. *The Oxford Handbook of Computational Linguistics*, chapter in R. Mitkov (ed.) Parsing, pages 233–248. Oxford University Press, 2003.

[CFMPN10] F. Claude, A. Fariña, M.A. Martínez-Prieto, and G. Navarro. Compressed q-gram Indexing for Highly Repetitive Biological Sequences. In *Proc. 10th International Conference on Bioinformatics and Bioengineering (BIBE-2010)*, 2010.

[Che96] S.F. Chen. *Building Probabilistic Models for Natural Language*. PhD thesis, Harvard University, MA, USA, 1996.

[CK08] E. S. Conley and S. T. Klein. Using alignment for multilingual text compression. *International Journal of Foundations of Computer Science*, 19(1):89–101, 2008.

[CLL+05] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

[CM05] J.S. Culpepper and A. Moffat. Compact Set Representation for Information Retrieval. In *12th String Processing and Information Retrieval Symposium (SPIRE 2005)*, pages 1–12, 2005.

[CN07] F. Claude and G. Navarro. A fast and compact Web graph representation. In *14th International Symposium on String Processing and Information Retrieval (SPIRE 2007)*, pages 105–116, 2007.

[CT91] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[CT95] J.G. Cleary and W.J. Teahan. Experiments on the zero frequency problem. In *Data Compression Conference (DCC 1995*, page 480, 1995.

[CT98] B. Chapin and S. R. Tate. Higher compression from the burrows-wheeler transform by modified sorting. In *Data Compression Conference (DCC 1998)*, page 532, 1998.

[Cul08] J.S. Culpepper. *Efficient Data Representations for Information Retrieval*. PhD thesis, Department of Computer Science and Software Engineering, University of Melbourne, Australia, 2008.

[CW79] B. Commentz-Walter. A String Matching Algorithm Fast on the Average. In *6th Colloquium on Automata, Languages and Programming*, pages 118–132, 1979.

[CW84a] J.G. Cleary and I.H. Witten. A comparisson of enumerative and adaptive codes. *IEEE Transactions on Information Theory*, 30(2):306–315, 1984.

[CW84b] J.G. Cleary and I.H. Witten. Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, 32(4):396–402, April 1984.

[CW02] A. Cannane and H.E. Williams. A General-Purpose Compression Scheme for Large Collections. *ACM Transactions on Information Systems*, 20(3):329–355, 2002.

[CW03] M. Carl and A. Way. *Recent Advances in Example-Based Machine Translation*. Springer, 2003.

[Die05] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 3 edition, 2005.

[dMNZ97] E. de Moura, G. Navarro, and N. Ziviani. Indexing compressed text. In *4th South American Workshop on String Processing*, pages 95–111, 1997.

[dMNZBY00] E. de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast and Flexible Word Searching on Compressed Text. *ACM Transactions on Information Systems*, 18(2):113–139, 2000.

[Dro02] A. Drozdek. *Elements of Data Compression*. Thomson Learning, 2002.

[Eli75] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.

[ES05] J. Eisner and N.A. Smith. Parsing with soft and hard constraints on dependency length. In *International Workshop on Parsing Technologies (IWPT)*, pages 30–41, 2005.

[Fan49] R.M. Fano. The Transmission of Information. Technical Report 65, Research Laboratory of Electronics, MIT, Cambridge, MA, 1949.

[Far05] A. Fariña. *New Compression Codes for Text Databases*. PhD thesis, Departamento de Computación, Universidade da Coruña, España, 2005.

[Fen94] Peter M. Fenwick. A New Data Structure for Cumulative Frequency Tables. *Software Practice & Experience*, 24(3):327–336,, 1994.

[FNP08] A. Fariña, G. Navarro, and J.R. Paramá. Word-Based Statistical Compressors as Natural Language Compression Boosters. In *Data Compression Conference (DCC 2008)*, pages 162–171, 2008.

[GB94] P. C. Gutmann and T. C. Bell. A hybrid approach to text compression. In *Data Compression Conference (DCC 1994)*, pages 225–233, 1994.

[GC93] W. A. Gale and K. W. Church. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(1):75–102, 1993.

[GF04] David A. Grossman and Ophir Frieder. *Information Retrieval: Algorithms and Heuristics*. Springer, 2004.

[GGMN05] R. González, S. Grabowski, V. Makinen, and G. Navarro. Practical implementation of rank and select queries. In *4th Workshop on Efficient and Experimental Algorithms (WEA 2005)*, pages 27–38, 2005.

[Gol66] S.W. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 12:399–401, 1966.

[HC92] R.N. Horspool and G.V. Cormack. Constructing word-based text compression algorithms. In *Data Compression Conference (DCC 1992)*, pages 62–81, 1992.

[Hea78] H.S. Heaps. *Information Retrieval - Computational and Theoretical Aspects*. Academic Press, 1978.

[HL90] D.S. Hirschberg and D.A. Lelewer. Efficient Decoding of Prefix Codes. *Communications of the ACM*, 4(33):449–459, 1990.

[Hoa62] C.A.R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–15, 1962.

[Hor80] R.N. Horspool. Practical fast searching in strings. *Software Practice & Experience*, 10(6):501–506, 1980.

[HT84] D. Harel and R.E. Tarjan. Fast algorithms for finding nearest common ancestors. *Siam Journal on Computing*, 13(2):338–355, 1984.

[Huf52] D.A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[HV92] P.G. Howard and J.S. Vitter. Practical Implementations of Arithmetic Coding. In J. A. Storer, editor, *Image and Text Compression*, pages 85–112. Kluwer Academic, 1992.

[HV94] P.G. Howard and J.S. Vitter. Arithmetic coding for data compression. Technical Report DUKE–TR–1994–09, Duke University, 1994.

[IV98] N. Ide and J. Véronis. Word Sense Disambiguation: The State of the Art. *Computational Linguistics*, 24(1):1–41, 1998.

[JN84] N. Jayant and P. Noll. *Digital Coding of Waveforms*. Prentice–Hall, 1984.

[KdM90] R. Kuhn and R. de Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, 1990.

[KF67] H. Kucera and W. N. Francis. *Computational analysis of present-day American English*. Brown University Press, 1967.

[KM98] H. Kruse and A. Mukherjee. Preprocessing text to improve compression ratios. In *Data Compression Conference (DCC 1998)*, page 556, 1998.

[KMP77] D. E. Knuth, J. H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[Koe05] P. Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of the Tenth Machine Translation Summit*, pages 79–86, 2005.

[Kou95] W. Kou. *Digital Image Compression: Algorithms and Standards*. Kluwer Academic Publishers, 1995.

[Kra49] L.G. Kraft. A device for quantizing, grouping and coding amplitude modulated pulses. Master's thesis, MIT, 1949.

[KY00] J.C. Kieffer and E. Yang. Grammar based codes: A new class of universal lossless source codes. *IEEE T.Inform.Theory*, 46:2000, 2000.

[Lar99] N. J. Larsson. *Structures of String Matching and Data Compression*. PhD thesis, Department of Computer Science, Lund University, Suecia, 1999.

[LM00] N.J. Larsson and A. Moffat. Offline Dictionary-Based Compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.

[LM07] M. Liddell and A. Moffat. Incremental calculation of minimum-redundancy length-restricted codes. *IEEE Transactions on Communications*, 55(3):427–435, 2007.

[Lop08] A. Lopez. *Machine Translation by Pattern Matching*. PhD thesis, University of Maryland, USA, 2008.

[Man53] B. Mandelbrot. An information theory of the statistical structure of language. *Communication Theory*, pages 503–512, 1953.

[Man97] U. Manber. A Text Compression Scheme that allows Fast Searching Directly in the Compressed File. *ACM Transactions on Information Systems*, 15(2):124–136, 1997.

[Mar79] G.N.N. Martin. Range Encoding: an Algorithm for Removing Redundancy from a Digitized Message. In *Video and Data Recording Conference*, 1979.

[McM56] B. McMillan. Two inequalities implied by unique decipherability. *Institute of Radio Engineers Transactions on Information Theory*, IT-2:115–116, 1956.

[Mel01] I. D. Melamed. *Empirical methods for exploiting parallel texts*. MIT Press, 2001.

[MI05] A. Moffat and R.Y.K. Isal. Word-based text compression using the Burrows-Wheeler transform. *Information Processing & Management*, 41(5):1175–1192, 2005.

[MNW98] A. Moffat, R.M. Neal, and I.H. Witten. Arithmetic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, 1998.

[Mof89] A. Moffat. Word-based text compression. *Software Practice & Experience*, 19(2):185–198, 1989.

[Mof90] A. Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921, 1990.

[Mof99] A. Moffat. An Improved Data Structure for Cumulative Probability Tables. *Software Practice & Experience*, 29(7):647–659, 1999.

[Mon01] M.A. Montemurro. Beyond the Zipf-Mandelbrot law in quantitative linguistics. *Physica A: Statistical Mechanics and its Applications*, 300(3-4):567–578, 2001.

[MPAdlF08] M. A. Martínez-Prieto, J. Adiego, and P. de la Fuente. Un modelo para el análisis y explotación de información cognitiva en repositorios documentales. In *XIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2008)*, pages 75–86, 2008.

[MPAdlFF10] M. A. Martínez-Prieto, J. Adiego, P. de la Fuente, and J.D. Fernández. High-Order Text Compression on Hierarchical Edge-Guided. In *Data Compression Conference (DCC 2010)*, 2010. 543.

[MPASM+09] M. A. Martínez-Prieto, J. Adiego, F. Sánchez-Martínez, P. de la Fuente, and R. C. Carrasco. On the use of word alignments to enhance bitext compression. In *Data Compression Conference (DCC 2009)*, page 459, 2009.

[MS05] R. Mihalcea and M. Simard. Parallel texts. *Natural Language Engineering*, 11(3):239–246, 2005.

[MSWB94] A. Moffat, N. Sharman, I.H. Witten, and T.C. Bell. An empirical evaluation of coding methods for multi-symbol alphabets. *Information Processing & Management*, 30(6):791–804, 1994.

[MT97] A. Moffat and A. Turpin. On the implementation of minimum-redundancy prefix codes. *IEEE Transactions on Communications*, 45(10):1200–1207, 1997.

[MT02] A. Moffat and A. Turpin. *Compression and Coding Algorithms*. Kluwer Academic Publishers, 2002.

[NMB92] C. G. Nevill-Manning and T. C. Bell. Compression of parallel texts. *Information Processing & Management*, 28(6):781–794, 1992.

[NMI97] C.G. Nevill-Manning and I.H.Witten. Compression and Explanation Using Hierarchical Grammars. *The Computer Journal*, 40(2/3):103–116, 1997.

[NR02] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings – Practical online search algorithms for texts and biological sequences*. Cambridge University Press, 2002.

[NS06] V. Nastase and S. Szpakowicz. Matching Syntactic-Semantic Graphs for Semantic Relation Assignment. In *Graph-based Algorithms for Natural Language Processing*, 2006.

[ON03] F. J. Och and H. Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

[Pas76] R. Pascoe. *Source coding algorithms for fast data compression*. PhD thesis, Stanford University, CA, USA, 1976.

[Ric79] R.F. Rice. Some practical universal noiseless coding techniques. Technical Report 79-22, Jet Propulsion Laboratory, CA, USA, 1979.

[Ris76] J. Rissanen. Generalised kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20:198–203, 1976.

[RL81] J. Rissanen and G.G. Langdon. Universal modeling and coding. *IEEE Transactions on Information Theory*, IT-27(1):12–23, 1981.

[Rob82] M.G. Roberts. *Local Order Estimating Markovian Analysis for Noisless Source Coding and Authorship Identification*. PhD thesis, Stanford University, CA, USA, 1982.

[Rub76] F. Rubin. Experiments in text file compression. *Communications of the ACM*, 19(11):617–623, 1976.

[Sak05] H. Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *Journal of Discrete Algorithms*, 3:416–430, 2005.

[Sal07] D. Salomon. *Variable-Length Codes for Data Compression*. Springer, 2007.

[Say02] K. Sayood. *Lossless Compression Handbook*. Academic Press, 2002.

[Sch98] M. Schindler. A Fast Renormalisation for Arithmetic Coding. In *Data Compression Conference (DCC 1998)*, page 572, 1998.

[SGD05] P. Skibiński, S. Grabowski, and S. Deorowicz. Revisiting dictionary-based compression. *Software Practice & Experience*, 35(15):1455–1476, 2005.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:398–403, 1948.

[Sha51] C. E. Shannon. Prediction and entropy of printed english. *Bell Systems Technical Journal*, 30:50–64, 1951.

[SK64] E. S. Schwartz and B. Kallick. Generating a Canonical Prefix Encoding. *Communications of the ACM*, 7:166–169, 1964.

[Smi02] M.A. Smirnov. Techniques to enhance compression of texts on natural languages for lossless data compression methods. In *V session of post-graduate students and young scientists of St. Petersburg*, 2002.

[SS82] J.A. Storer and T.G. Szymanski. Data Compression Via Textual Substitution. *Journal of the ACM*, 29(4):928–951, 1982.

[Sun90] D.M. Sunday. A very fast substring search algorithm. *Commun. ACM*, 33(8):132–142, 1990.

[SZM04] W. Sun, N. Zhang, and A. Mukherjee. A dictionary-based multi-corpora text compression system. In *Data Compression Conference (DCC 2004)*, page 448, 2004.

[TC96] W. J. Teahan and J.G. Cleary. The Entropy of English Using PPM-based Models. In *Data Compression Conference (DCC 1996)*, pages 53–62, 1996.

[TC97a] W.J. Teahan and J.G. Cleary. Models of English Text. In *Data Compression Conference (DCC 1997)*, pages 12–21, 1997.

[TC97b] W.J. Teahan and J.G. Cleary. Unbounded Length Contexts for PPM. *The Computer Journal*, 40(2/3):67–75, 1997.

[Tea97] W. J. Teahan. *Modelling English Text*. PhD thesis, Department of Computer Science, University of Waikato, Nueva Zelanza, 1997.

[TICH98] W. J. Teahan, S. Inglis, J.G. Cleary, and G. Holmes. Correcting English text using PPM models. In *Data Compression Conference (DCC 1998)*, pages 289–298, 1998.

[TM97] A. Turpin and A. Moffat. Constructing word-based text compression algorithms. In *20th Australian Computer Science Conference*, pages 1–8, 1997.

[VNT96] S. Vogel, H. Ney, and C. Tillmann. HMM-based word alignment in statistical translation. In *16th International Conference on Computational Linguistics (COLING'96)*, pages 836–841, 1996.

[Wan03] R. Wan. *Browsing and Searching Compressed Documents*. PhD thesis, Department of Computer Science and Software Engineering, University of Melbourne, Australia, 2003.

[WB90] I.H. Witten and T.C. Bell. Source Models for Natural Language Text. *International Journal of Man-Machine Studies*, 32(5):545–579, 1990.

[WB91] I.H. Witten and T.C. Bell. The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, 1991.

[Wel84] T.A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984.

[WMB99] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes*. Morgan Kaufmann Publishers, Inc., 1999.

[WNC87] I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic coding for data compression. *Communications of ACM*, 30(6):520–540, 1987.

[WZ99] H.E. Williams and J. Zobel. Compressing Integers for Fast File Access. *The Computer Journal*, 42:193–201, 1999.

[Yur98] D. Yuret. *Discovery of linguistic relations using lexical attraction*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, USA, 1998.

[ZdMNBY00] N. Ziviani, E. de Moura, G. Navarro, and R. Baeza-Yates. Compression: A key for next-generation text retrieval systems. *IEEE Computer*, 33(11):37–44, 2000.

[Zip49] G. K. Zipf. *Human behavior and the principle of least effort*. Addison-Wesley, 1949.

[ZL77] J. Ziv and A. Lempel. An universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[ZL78] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[ZM95] J. Zobel and A. Moffat. Adding compression to a full-text retrieval system. *Software & Practice and Experience*, 25(8):891–903, 1995.

[ZWH01] J. Zobel, H.E. Williams, and S. Heinz. In-memory Hash Tables for Accumulating Text Vocabularies. *Information Processing Letters*, 80(6):271–277, 2001.