Búsqueda Informada

Algoritmos primero el mejor

Algoritmos de búsqueda local

Algoritmos primero el mejor

Búsqueda primero el mejor
Búsqueda Voraz
A*

Algoritmos primero el mejor

- Familia de algoritmos de búsqueda en grafos, informados
- Se caracterizan por el uso de una función heurística para guiar la búsqueda, con mínimo global en meta
  - f : U $\rightarrow$R, U=EE $\cup$ Caminos EE, R reales
  - Criterio de expansión de nodos: expandir primero el nodo con menor valor de f
- Si no imponemos restricción adicional a f, Búsqueda primero el mejor

Búsqueda primero el mejor: bpm

1. Si $\mathbf{n_0}$ es meta, fin con éxito, devolviendo $\mathbf{n_0}$
2. **Abiertos** $\leftarrow$ ($\mathbf{n_0}$); **Cerrados** $\leftarrow$ ( )
3. Si **Abiertos** = ( ), fin devolviendo fallo
4. **n** $\leftarrow$ primer elemento de **Abiertos**; eliminar **n** de **Abiertos** y llevarlo a **Cerrados**; **Suc** $\leftarrow$ ( )
5. expandir **n**, colocando sus hijos en **Suc**, como hijos de **n**
6. Si alguno de los hijos de **n** es un nodo meta, fin con éxito, devolviendo el camino
7. eliminar de **Suc** cualquier nodo cuyo estado ya esté asociado a algún nodo de **Abiertos** o **Cerrados**
8. colocar los nodos de **Suc** en **Abiertos**
9. Reordenar **Abiertos** según valores crecientes de f(**n**)
10. Ir a 3

Comportamiento bpm

- No es completo
- No es admisible
- Complejidad espacial, temporal, exponencial

Búsqueda Voraz (Greedy Search)

- Utiliza funciones heurísticas del estado
- Esta función debe de informar sobre la calidad del estado:
  - Coste de alcanzar la solución desde el estado actual
  - Independiente del coste del camino hasta el estado actual
- Estas funciones heurísticas se denotan por *h*

Búsqueda Voraz

1. Si $\mathbf{n_0}$ es meta, fin con éxito, devolviendo $\mathbf{n_0}$
2. **Abiertos** ← ($\mathbf{n_0}$); **Cerrados** ← ( )
3. Si **Abiertos** = ( ), fin devolviendo fallo
4. **n** ← primer elemento de **Abiertos**; eliminar **n** de **Abiertos** y llevarlo a **Cerrados**; **Suc** ← ( )
5. expandir **n**, colocando sus hijos en **Suc**, como hijos de **n**
6. Si alguno de los hijos de **n** es un nodo meta, fin con éxito, devolviendo el camino
7. eliminar de **Suc** cualquier nodo cuyo estado ya esté asociado a algún nodo de **Abiertos** o **Cerrados**
8. colocar los nodos de **Suc** en **Abiertos**
9. Reordenar **Abiertos** según valores crecientes de h(**n**)
10. Ir a 3

Comportamiento búsqueda voraz

- Como primero el mejor
- Principal inconveniente: no considera el coste de los caminos que encuentra
  - Puede explorar caminos de longitud infinita

Algoritmos A*

- Idea: penalizar el coste del camino para incluir un “componente en anchura”
- f=g+h con
  - g coste camino mínimo encontrado por el algoritmo hasta el nodo
  - h estimación coste camino mínimo del estado a la meta
- g estima g*: coste del camino mínimo hasta nodo ($g \geq g^*$)
- h estima h*: coste del camino mínimo del nodo a meta
- Si h es minorante de h* ($h \leq h^*$), Algoritmo A*

Algoritmo A*

1. **Abiertos** ← (**n**$_0$); **Cerrados** ← ( )
2. Si **Abiertos** = ( ), fin devolviendo fallo
3. **n** ← primer elemento de **Abiertos**; eliminar **n** de **Abiertos** y llevarlo a **Cerrados**; **Suc** ← ( )
4. Si **n** es un nodo meta, fin con éxito devolviendo el camino
5. expandir **n**, colocando sus hijos en **Suc**, como hijos de **n**
6. Para cada nodo **q** de **Suc**
   - 6.1. Calcular g(**q**) ← g(**n**) + coste(**n**,**q**)
   - 6.2. Si **q** no está ni en **Abiertos** ni en **Cerrados**, calcular f(**q**) ← g(**q**) + h(**q**) y añadir **q** a **Abiertos** asignando f(**q**) y g(**q**)
   - 6.3. Si **q** estaba en **Abiertos** o en **Cerrados** (**q**$_0$), comparar el valor de g(**q**) con g(**q**$_0$)
     1. Si g(**q**$_0$) <= g(**q**), descartar **q**
     2. Si g(**q**$_0$) > g(**q**), colocar **n** como nuevo padre de **q**$_0$, asignarle f(**q**) = g(**q**) + h(**q**$_0$) y descartar **q**
     3. Si g(**q**$_0$) > g(**q**) y **q**$_0$ está en **Cerrados**, eliminarle de **Cerrados** y llevarle a **Abiertos**
7. Reordenar **Abiertos** según valores crecientes de f(**n**)
8. Ir a 2

Propiedades formales de A*

- Teorema 1: A* es completo en grafos localmente finitos
- Teorema 2: A* es admisible en grafos localmente finitos

- Def. Heurística mejor informada

  $h_2(n)$ mejor informada $h_1(n)$

  ambas minorantes, $h_2(n) > h_1(n)$ en nodos no terminales
- Def. algoritmo A* más informado
- Def. algoritmo dominante
- Teorema 3. Sean A_1^* y A_2^* dos algoritmos A* para el mismo problema. Si A_2^* está mejor informado que A_1^*, A_2^* domina a A_1^*

Comportamiento A*

- Completo, Admisible
- Óptimamente eficiente
  - Algoritmo admisible que genera menos nodos para una heurística dada
- Pero coste exponencial salvo

  $|h(n) - h^*(n)| \leq O(\log h^*(n))$

  - En la práctica, el error es al menos proporcional al coste del camino y el coste espacial es exponencial

Variantes A*

- Utilizando muy poca memoria
  - A*DI, A* descenso iterativo
  - BRPM, búsqueda recursiva primero el mejor
- Utilizando memoria disponible
  - A*M, A* con memoria acotada
  - A*Ms, A*M simplificado

Algoritmos de búsqueda local

Mejora iterativa

Métodos de población

Algoritmos de búsqueda local

- Sólo mantienen el estado(s) actual(es)
  - Régimen de control irrevocable
- Especialmente apropiados cuando no interesa el camino
  - Problemas de configuración: 8-reinas, VLSI
- Problemas de optimización
  - Viajante de comercio, funciones
- Ventajas
  - Usan muy poca memoria
  - Pueden alcanzar soluciones razonables en espacios de estado grandes o infinitos, poco adecuados para los algoritmos que mantienen los caminos
- Inconvenientes
  - Derivados de su carácter local

Mejora iterativa

- Solo mantienen el estado actual
  - Algoritmos con régimen de control irrevocable
- Trabajan introduciendo modificaciones en una configuración inicial para mejorar su calidad
    - Codificar estado de forma que contenga toda la información relevante de la solución
- Utilizan función de evaluación heurística con máximo (mínimo) global en soluciones
- Dos familias
  - Escalada: siempre dirección máximo (ascensión de colinas, hill-climbing)
  - Temple simulado: permiten empeoramiento (simulated annealing)

Método de escalada

- Un único estado
- Función de evaluación con máximo absoluto en solución óptima
- Se mueve siempre en la dirección en que mejora la función de evaluación
  - Si no puede, termina con fallo

Método de escalada

función ESCALADA(*problema*) **returns** un estado
entrada: *problema*, un problema
static: *actual*, un nodo
siguiente, un nodo
begin
actual ← HACER-NODO(ESTADO-INICIAL[*problema*])
loop do
siguiente ← sucesor de *actual* con máximo valor
if VALOR(*siguiente*) ≤ VALOR(*actual*) **then return**(*actual)*
actual ← *siguiente*
endloop
end

Limitaciones escalada

- Máximos locales
- Llanuras
- Crestas

Comportamiento escalada

- No es completo ni admisible
  - Puede encontrar soluciones razonablemente buenas
- Complejidad espacial: K (bytes para almacenar estado)
- Complejidad temporal: peor caso, exponencial
  - En la práctica, suele terminar con pocas iteraciones
- Comportamiento 8-reinas
  - Estado: tablero con 8 reinas, una por columna
  - Operadores: mover una reina a otra casilla de la misma columna
  - Fallo 86%, éxito 14%
  - Rápido. En media
    - 4 pasos si éxito
    - 3 pasos si fallo

Variantes de escalada

- Escalada estocástico
  - Escoge aleatoriamente entre los hijos que mejoran la función de evaluación
  - Puede mejorar solución encontrada con escalada
- Permitir movimientos laterales (función de evaluación constante)
  - Para atravesar llanuras. Ciclos
- Escalada con reinicio aleatorio
  - Sucesivas iteraciones método escalada
  - Estados iniciales generados aleatoriamente
  - Conserva el mejor estada encontrado
  - Terminación: num. máximo de iteraciones, no hay mejora significativa, etc.

Comportamiento escalada con reinicio aleatorio

- Completo en el límite
  - Si nº reinicios tiende a infinito, eventualmente generará la solución aleatoriamente
- Suele mejorar solución encontrada por escalada
- Inicios medios requeridos: 1/p (con p probabilidad de éxito de escalada)
- 8-reinas
  - p~0.14; reinicios ~ 7

Enfriamiento simulado

- Introduce un componente aleatorio sobre el método de escalada
- Permite moverse en direcciones en que empeora la función de evaluación
  - Permite salir de máximos locales
- Inicialmente, con mucha facilidad
- En las etapas finales, con más dificultad
- Escoge el sucesor aleatoriamente
- Los movimientos que empeoran se aceptan con probabilidad que disminuye
  - Con el empeoramiento
  - Temperatura (tiempo)

Temple simulado: inspiración física

- Inspiración física en el proceso de “temple”
  - Primero calentar: movimiento aleatorio de átomos y moléculas que permite salir de estados físicos correspondientes a mínimos locales (energía)
  - Enfriamiento paulatino para alcanzar estados de mínima energía
  - Cuanto más lento, más fácil llegar a mínimos globales (buscando, en algún momento, que el movimiento permita salir de mínimos locales pero no de los globales)

Enfriamiento simulado

función ENFRIAMIENTO-SIMULADO(*problema*, *schedule*) **returns** un estado

entrada: *problema*, un problema
schedule, una aplicación de tiempo en "temperatura"

static: *actual*, un nodo
siguiente, un nodo
T, una "temperatura"

begin
actual ← HACER-NODO(ESTADO-INICIAL[*problema*])
for t ← 1 **to** ∞ **do**
T ← *schedule*[t]
if *T* = 0 **then return**(*actual)*
siguiente ← un sucesor de *actual* seleccionado aleatoriamente
ΔE ← VALOR[*siguiente*] – VALOR[*actual*]
if ΔE > 0 **then** *actual* ← *siguiente*
else *actual*← *siguiente* con probabilidad $e^{\Delta E/T}$
enddo
end

Comportamiento temple simulado

- Útil para aproximar el máximo global en espacio de estado grandes
- El algoritmo encuentra óptimo global con probabilidad próxima a 1 si T
  - Monótona decreciente con t
  - Disminución “lenta y suave”
- No es completo ni admisible. Complejidad temporal peor caso: exponencial
  - En la práctica, en el peor caso, termina con T=0. Luego la función Schedule determina la complejidad temporal.

Métodos de población

- Búsqueda en haz local
- Algoritmos genéticos

Métodos de población

- Métodos de búsqueda local
  - No mantienen el camino, solo estados
- Mantienen varios candidatos
  - Población
- Dos variantes
  - Búsqueda en haz local
    - Variante de métodos de mejora iterativa
    - Reproducción asexuada
  - Algoritmos genéticos
    - Reproducción sexuada

Búsqueda en haz

- Búsqueda en haz: algoritmos de búsqueda que mantienen en memoria un número limitado de K nuevos sucesores
  - Mantienen el camino: variante de primero el mejor
  - Mantienen el estado: variante de mejora iterativa (búsqueda en haz local)

Búsqueda en haz local

- Parte de *K* estados generados aleatoriamente
- Genera los hijos de todos los estados
- Conserva los *K* estados con mejor valor de la función de evaluación heurística

Búsqueda en haz local

función BUSQUEDA-HAZ-LOCAL(*problema*, *k*) **returns** una lista de estados
entrada: *problema*, un problema
k, tamaño del haz
static: siguiente, *sucesores*, *población*, listas de nodos
begin
población ← HACER-LISTA-NODOS(ESTADOS-INICIALES-ALEATORIOS(*problema,k*))
while (not CondicionTerminación**) do**
siguiente ← ∅
for each *nodo* ∈ *población* **do**
sucesores ← HACER-LISTA-NODOS(EXPANDIR(*nodo*))
siguiente ← *siguiente* ∪ *sucesores*
end
población ← *k* mejores elementos de *siguiente*
endwhile
return(*población*)
end

Comportamiento búsqueda en haz local

- No garantiza solución óptima
- Generalmente, mejor comportamiento que escalada con reinicio aleatorio k-veces: se comparte información útil entre los elementos de la población
- Mejora: búsqueda en haz estocástico
  - Se eligen k sucesores aleatoriamente con probabilidad creciente con la heurística
  - Cierta similitud con la evolución natural de poblaciones