Colored Point Set Matching Under Noise in 3D

Yago Diez [*] J. Antoni Sellarès [*]

Abstract

Let $\mathcal{A}$ and $\mathcal{B}$ be two colored points sets in $\mathcal{R}^3$, with $|\mathcal{A}| \leq |\mathcal{B}|$. We propose a process for determining matches, in terms of the *bottleneck* distance, between $\mathcal{A}$ and subsets of $\mathcal{B}$ under color preserving rigid motion, assuming that the position of all colored points in both sets contains a certain amount of "noise". Our algorithm consists of two main stages. In the first one we generate all, up to a certain equivalence, possible motions that bring $\mathcal{A}$ close to some subset $\mathcal{B}'$ of $\mathcal{B}$ and seek for a matching between sets $\mathcal{A}$ and $\mathcal{B}'$. To look for these possible matchings we use a bipartite matching algorithm that uses skip octrees for neighborhood queries. Additionaly, we also present a lossless filtering algorithm to discard those subsets of $\mathcal{B}$ where there can not appear any matches and thus, improve the efficiency of our process. This algorithm determines a number of *c*andidate zones which are regions that contain a subset $\mathcal{S}$ of $\mathcal{B}$ such that $\mathcal{A}$ may match one or more subsets $\mathcal{B}'$ of $\mathcal{S}$. We use a compressed octree to have easy access to the subsets of $\mathcal{B}$ related to candidate zones and store geometric information that is used by the lossless filtering algorithm in each octree node.

1 Introduction

Protein molecules possess unique three-dimensional structures, defined by their amino-acid sequence, that play a very important role in molecular biology. Many functional properties of proteins have been found to depend on some typical parts of their three-dimensional structure. Typically a protein molecule is modelled as a set of balls in $\mathbb{R}^3$, each representing an atom. Given a substructure understood as a small collection of atoms, representing a secondary structure subunit or any other significant part of a protein (as some significant union of such secondary structures, called *motifs*), the *protein substructure detection problem* consists in determining whether the substructure exists in a protein molecule [7]. Here it is important to notice that the types of atoms conforming a protein are a finite set, so if we consider the centers of the atoms involved and colors corresponding to the types of atoms we can see the problem as a colored point set matching problem. Since atom positions are fuzzy due to the finite precision of measuring devices, it is impractical to consider an exact match between two atoms. Therefore, atoms are considered superimposed if the distances between their centers are less than some positive constant, the *center location error* (ϵ), small enough compared to inter-center distances. With this extra consideration we can define our problem in terms of a Noisy Colored Point Set Matching problem. Notice that if all the points have the same color we get, as a particular case, the widely known Noisy Point Set Matching problem. Another aspect that we want to highlight is that the sets to be matched do not have the same cardinality, so the objective is to match one of the sets to a subset of the other (this is also known as partial matching). Finally, bearing in mind that in our motivational problems the correspondences between the colored points to be matched are required to be one-to-one, we will use the *bottleneck* distance.

1.1 Previous Results

The study of the Noisy Point Set Matching in 2D problem was initiated by Alt *et al.* [2] who presented an exact $O(n^8)$ time algorithm for solving the problem for two sets $\mathcal{A}, \mathcal{B}$ of cardinality n. Combining Alt et alt. algorithm with the techniques by Efrat *et al.* [8] the time can be reduced to $O(n^7 \log n)$.

The 3D version of the problem is much less explored than its two dimensional counterpart. The only algorithms to the best of our knowledge are: an algorithm by Ambühl, Chakraborty and Gartner presented

[*]Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain. Email: `{ydiez,sellares}@ima.udg.es`. Partially supported by the Spanish Ministerio de Educación y Ciencia under grant TIN2004-08065-C02-02.

in [1] that suffered from a computational cost of $O(n^{32.5})$ and a $O(n^{13+5/6+\epsilon})$ result presented in [5] that also uses techniques by [8]. Biswas and Chakraborty [3] present an algorithm for solving this problem exactly in the particular case of protein molecules, that uses geometric properties of the carbon chains that constitute them that is claimed to be fast enough to be useful in practice.

2 Problem Definition

Let $P(q,r)$ represent the colored point $q \in \mathbb{R}^3$ with associated color r. Given a real number $\epsilon \geq 0$, we say that two colored points $A = P(a,r)$,and $B = P(b,s)$ *approximately match* when $r = s$ and $\tilde{d}(A,B) = d(a,b) \leq \epsilon$, where d denotes the Euclidean distance. For the sake of concretion, from now on we will use the term *match* instead of *approximately match*.

Let $\mathcal{D}$, $\mathcal{S}$ be two colored points sets of the same cardinality. A *color preserving bijective mapping* $f : \mathcal{D} \to \mathcal{S}$ maps each colored point $A = P(a,r) \in \mathcal{D}$ to a distinct and unique colored point $f(A) = P(b,s) \in \mathcal{S}$ so that $r = s$. Let $\mathcal{F}$ be the set of all color preserving bijective mappings between $\mathcal{D}$ and $\mathcal{S}$. The *bottleneck distance* between $\mathcal{D}$ and $\mathcal{S}$ is defined as:

$$d_b(\mathcal{D},\mathcal{S}) = \min_{f\in\mathcal{F}} \max_{A\in\mathcal{D}} \tilde{d}(A, f(A))\,.$$

The **Noisy Colored Point Set Matching (NCPSM)** problem can be formulated as follows. Given two Colored Points sets $\mathcal{A}$, $\mathcal{B}$, $|\mathcal{A}| = n$, $|\mathcal{B}| = m$, $n \leq m$, and $\epsilon \geq 0$, determine all rigid motions τ for which there exists a subset $\mathcal{B}'$ of $\mathcal{B}$ such that $d_b(\tau(\mathcal{A}),\mathcal{B}') \leq \epsilon$. We define $\tau(P(a,r))$ as $P(\tau(a),r)$ and $\tau(\mathcal{A})$ as $\{\tau(P(a,r)) \mid P(a,r) \in \mathcal{A}\}$

If τ is a solution to the **NCPSM** problem, every colored point of $\tau(\mathcal{A})$ matches to a distinct and unique colored point of $\mathcal{B}'$ of the same color, and we say that $\mathcal{A}$ and the subset $\mathcal{B}'$ of $\mathcal{S}$ are *noisy congruent.* For the sake of clarity, through the rest of the paper we will just mention the colors associated to points only when necessary, thus we will speak about $a \in \mathcal{A}$ instead of $P(a,r) \in \mathcal{A}$.

3 NCPSM Problem Solving Algorithm

We present an algorithm to solve the **NCPSM** problem that extends the algorithm for the 2D version of the problem presented in [6]. The algorithm consists on two parts called "enumeration" and "testing". In the "enumeration" part we group all possible motions of set $\mathcal{A}$ in equivalence classes to make their handling possible and then generate a representative in each class. In the "testing" part we run a bipartite matching algorithm for every resulting transformation of set $\mathcal{A}$ in order to determine if it matches any subset of $\mathcal{B}$.

3.1 Enumeration

Generating every possible rigid motion that brings set $\mathcal{A}$ onto a subset of $\mathcal{S}$ is infeasible due to the continuous nature of movement. Following the 2D algorithm in [2], we partition the set of all rigid motions in equivalence classes in order to make their handling possible. Consider the arrangement of spheres $\mathcal{S}_\mathcal{B} = \{S(b,\epsilon)|b \in \mathcal{B}\}$. Any point $p \in \mathcal{R}^3$ belongs to the cell of the arrangement identified with the set of points $b \in \mathcal{B}$ that hold that $p \in B(b,\epsilon)$.

Definition 3.1. Two motions τ, μ are *equivalent* if and only if, for any colored point $a \in \mathcal{A}$ points $\tau(a)$ and $\mu(a)$ lie in the same cell of $\mathcal{S}_\mathcal{B}$.

The next Lemma ensures that a finite set of motions suffices for the purpose of finding all the solutions of the **NCPSM** problem.

Lemma 3.2. *Any rigid motion μ that is a solution to the* **NCPSM** *problem can be transformed to another equivalent motion μ' such that there exist three points $a_{i_1}, a_{i_2}, a_{i_3} \in \mathcal{A}$ whose images $\mu'(a_{i_1}), \mu'(a_{i_2}), \mu'(a_{i_3})$ lie on the spheres $S(b_{j_1},\epsilon), S(b_{j_2},\epsilon), S(b_{j_3},\epsilon)$ respectively, where $b_{j_1} = \mu(a_{i_1})$, $b_{j_2} = \mu(a_{i_2})$ and $b_{j_3} = \mu(a_{i_3})$.*

Proof. First we transform the set $\mu(\mathcal{A})$ by a translation T in any given direction until a point $p = T(\mu(a_{i_1}))$, $a_{i_1} \in \mathcal{A}$ belongs to $S(b_{j_1}, \epsilon)$, where $b_{j_1} = \mu(a_{i_1}) \in \mathcal{B}$. Next we rotate the translated set $T(\mu(\mathcal{A}))$ by the rotation R of axis the line passing through p and whose direction is the translation vector of T until a point $q = R(T(\mu(a_{i_2}))$, $a_{i_2} \in \mathcal{A})$, belongs to $S(b_{j_2}, \epsilon)$, where $b_{j_2} = \mu(a_{i_2}) \in \mathcal{B}$. Finally we rotate the resulting set $R(T(\mu(\mathcal{A})))$ by the rotation R' whose axis is the line through p and q until the point $r = R'(R(T(\mu(a_{i_3})))$, $a_{i_3} \in \mathcal{A})$, belongs to $S(b_{j_3}, \epsilon)$, where $b_{j_3} = \mu(a_{i_3}) \in \mathcal{B}$. In these conditions, the motion $\mu' = R' \circ R \circ T \circ \mu$ holds Lemma 3.2. □

Notice that, as a consequence of Lemma 3.2 we only need to consider those motions that bring three points in set $\mathcal{A}$ to the boundary of the noise regions of their corresponding points in the matching. In this section we present an algorithm that generates a representative for each of the equivalence classes of this motions. Each rigid motion τ of set $\mathcal{A}$ is determined by the image of any three points. We choose these points to be the ones that lie in the boundary of their corresponding points in the matching. Consequently, we consider each possible 6-tuple $a_{i_1}, a_{i_2}, a_{i_3}, b_{j_1}, b_{j_2}, b_{j_3}$ with $a_{i_1}, a_{i_2}, a_{i_3} \in \mathcal{A}$ and $b_{j_1}, b_{j_2}, b_{j_3} \in \mathcal{B}$. As the matching must preserve colors, we also demand the colors associated to points to be matched to be the same i.e. $P(a_{i_1}, r_1) \in \mathcal{A}$, $P(b_{i_1}, r_1) \in \mathcal{B}$, $P(a_{i_2}, r_2) \in \mathcal{A}$, $P(b_{i_2}, r_2) \in \mathcal{B}$ and finally, $P(a_{i_3}, r_3) \in \mathcal{A}$, $P(b_{i_3}, r_3) \in \mathcal{B}$.

Lemma 3.3. *Any given position of point $\tau(a_{i_1})$ in $S(b_{j_1}, \epsilon)$ leaves a degree of freedom for the positions of $\tau(a_{i_2})$ and $\tau(a_{i_3})$ in $S(b_{j_2}, \epsilon)$ and $S(b_{j_3}, \epsilon)$ respectively.*

Proof. For each possible position of point $\tau(a_{i_1})$, consider the geometric locus of all the points that belong to $S(b_{j_2}, \epsilon)$ and whose distance to $\tau(a_{i_1})$ is exactly $d(a_{i_1}, a_{i_2})$. This corresponds, in the general case to a circle $C_{a_{i_1}, a_{i_2}} \subset \mathbb{R}^3$ resulting from the intersection of $S(b_{j_2}, \epsilon)$ and $S(\tau(a_{i_1}), d(a_{i_1}, a_{i_2}))$. Consider also the geometric locus of all the points that belong to $S(b_{j_3}, \epsilon)$ and whose distance to $\tau(a_{i_1})$ is exactly $d(a_{i_1}, a_{i_3})$. Finally by choosing a point in one of the circles (determining, thus $\tau(a_{i_2})$) and imposing that $d(\tau(a_{i_2}), \tau(a_{i_3})) = d(a_{i_2}, a_{i_3})$, remains only a degree of freedom. □

The configuration space of our problem can be seen as a cube of side 2π where each dimension corresponds to one of the three angles that determine the associated rigid motion i.e. the two polar coordinates ϕ, ψ that determine the position of $\tau(a_{i_1})$ and the remaining angle θ corresponding to the point chosen in $C_{a_{i_1}, a_{i_2}}$. From now on, we will denote $\tau_{\phi\psi\theta}$ the rigid motion that corresponds to any given value of parameters ϕ, ψ and θ. Another key observation is that, for any given values of parameters ϕ, ψ, θ any couple of the remaining points $a_{i_h} \in \mathcal{A}$, $b_{i_l} \in \mathcal{B}$ defines three possible positions corresponding to the position of $\tau_{\phi\psi\theta}(a_{i_h})$ respect to the sphere $S(b_{i_l}, \epsilon)$ (in, out or in the surface). This three possible positions correspond to the values of ϕ, ψ and θ for which a_{i_h} and b_{i_l} may (or may not) be matched. In this way, given a 6-tuple $a_{i_1}, a_{i_2}, a_{i_3}, b_{j_1}, b_{j_2}, b_{j_3}$ and an additional couple a_{i_h}, b_{i_l}, we have a finite number of regions of $[0, 2\pi[^3$ that encode the information of when $\tau(a_{i_h})$ may be matched to b_{i_l} for a motion τ that brings $a_{i_1}, a_{i_2}, a_{i_3}$ to the boundary of the spheres $S(b_{j_1}, \epsilon), S(b_{j_2}, \epsilon), S(b_{j_3}, \epsilon)$, respectively.

Consequently, in searching for the possible matchings, it suffices to consider the collection of $(n-3)(m-3) \in O(nm)$ surfaces $\mathcal{S}$ of $[0, 2\pi[^3$ determined by the angles ϕ, ψ, θ such that $\tau_{\phi\psi\theta}(a_{i_h})$ belongs to the boundary of the sphere $S(b_{i_l}, \epsilon)$, $a_{i_h} \notin \{a_{i_1}, a_{i_2}, a_{i_3}\}$, $b_{jl} \notin \{b_{j_1}, b_{j_2}, b_{j_3}\}$. Since we only need to encode the adjacency relationship among 3-dimensional cells of the arrangement, first we compute the vertical decomposition of the arrangement and then we compute the adjacency relationship of cells in the decomposition [12]. The number of cells of the vertical decomposition of the arrangement of the $O(nm)$ surfaces in $\mathcal{S}$ is $O(n^2 m^2 \lambda_q(nm))$, where q is an integer constant depending on the maximum degree of the surfaces and $\lambda_s(k)$ is the maximum length of (k, s) Davenport-Schinzel sequences that is roughly linear in k for any constant s [13]. The vertical decomposition of the arrangement can be computed in randomized expected time $O((nm)^{3+\epsilon})$, using the random-sampling technique [4]. Putting it all together, what we need to determine in order to generate a representative in every equivalent class of motions is the arrangement of surfaces in the cube $[0, 2\pi[^3$ defined by all possible couples once fixed a 6-tuple and then iterate over the set of all 6-tuples. This takes $O(n^{6+\epsilon} m^{6+\epsilon})$ expected time and $O(n^5 m^5 \lambda_q(nm))$ space.

3.2 Testing

In this section we present an algorithm to test if each of the motions for set $\mathcal{A}$ generated in the Enumeration step matches some subset of set $\mathcal{B}$. Our algorithm uses the bipartite matching algorithm presented in [11]. For the implementation we adapt the ideas presented in [6] and [8] for the 2D case. We generate values for parameters ϕ, ψ and θ inside each of the regions of the cube $[0, 2\pi[^3$ defined in the Enumeration section and test sets $\tau_{\phi\psi\theta}(\mathcal{A})$ and $\mathcal{B}$ for matching. The first time we generate a *layered graph* [8] that contains information on proximity relations between the points in $\tau_{\phi\psi\theta}(\mathcal{A})$ and in $\mathcal{B}$ and run Hopcroft and Karp's [11] algorithm.

Whenever the testing part determines a matching of cardinality n we annotate the corresponding $\tau_{\phi\psi\theta}$ and proceed. Every time we consider a new region representing an equivalence class of motion we can update the matching by changing a single edge in the layered graph and finding a single augmenting path. If we add an edge to the layered graph, the matching increases by at most one edge. Therefore, we look for an augmenting path which contains the new edge. If we remove an edge from the layered graph, we need to search for a single augmenting path.

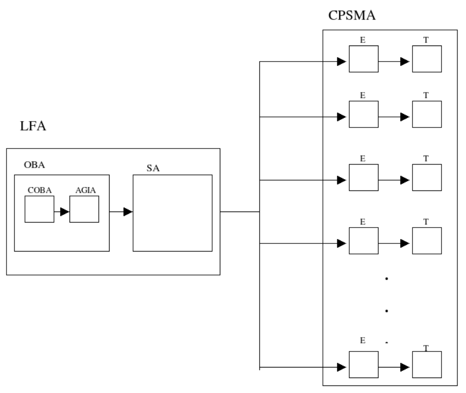
When searching for augmenting paths we need to perform efficiently two operations. a) **neighbor** $(D(\mathcal{T}), q)$: for a query point q in a data structure $D(\mathcal{T})$ that represents a point set $\mathcal{T}$, return a point in $\mathcal{T}$ whose distance to q is at most ϵ or $\emptyset$ if no such element exists. b) **delete**$(D(\mathcal{T}), s)$: deletes point s from $D(\mathcal{T})$. For our implementation we use the *skip octree*, a data structure that combines the best features of octrees and a skip lists [9]. The cost of building a skip octree for any $\mathcal{T} \subset \mathcal{B}$ with $|T| = n'$ is in $O(n' \log n')$. In the worst case, when $n' = m$, this computational cost is the same needed to build the data structure used in [8]. The asymptotic computational cost of the **delete** operation in $\mathcal{T}$'s skip octree is $O(\log n')$. The **neighbor** operation is used combined with the delete operation to prevent re-finding points. This corresponds to a range searching operation in a skip octree followed by a set of deletions. The range searching can be approximated in $O(\delta^{-2} \log n' + u)$ time, where u is the size of the output, for a small constant δ such that $\epsilon > \delta > 0$ [9]. The approximate range searching outputs some "false" neighbor points that can be detected in $O(1)$ time. We will denote $t(n, n')$ an upper bound on the amortized time of performing **neighbor** operation in $\mathcal{T}$'s skip octree. This yields a computational cost of $O(nt(n, n'))$ for finding an augmenting path.

In the worst case $t(n, n') \in O(n')$. However, if we assume that the amount of noise in set $\mathcal{A}$ data is "reasonable“ $t(n, n') \in O(\log n)$. As Hopcroft-Karp Bipartite matching algorithm ends in, at most $n^{1.5}$ steps, the total cost of the testing algorithm is bounded by $O(n^{1.5}t(n, n'))$. However, as we only need to compute the first layered graph and update it in constant time for the remaining cases, the total amortized cost drops to $O(nt(n, n'))$. Consequently, testing the whole $O(n^5 m^5 \lambda_q(nm))$ cells takes $O(n^6 m^5 \lambda_q(nm) t(n, n'))$ total time.

4 Lossless Filtering Preprocessing Step

In most applications the cardinals of the two sets involved in matching problems are dissimilar and $|\mathcal{A}| << |\mathcal{B}|$. The algorithm that we have presented until now, as well as the algorithms in the literature are designed for sets of roughly equal cardinalities so cannot take advantage of this situation if it occurs. In this section we present an algorithm that discretizes the **NCPSM** problem by turning it into a series of smaller instances of itself and then solves them using the algorithm already presented. To achieve the discretization that we are looking for we use a conservative strategy that discards those subsets of $\mathcal{B}$ where no match may happen and keep a number of zones where this matches may occur.

Our process consists of two main algorithms. The First one performs a Lossless Filtering step that discards those parts of $\mathcal{B}$ that may not contain matches and outputs a number of "candidate zones". Each of these zones contains a subset $\mathcal{S}$ of $\mathcal{B}$ such that $\mathcal{A}$ may approximately match one or more $\mathcal{B}' \subset \mathcal{S}$. The second algorithm solves the **NCPSM** problem between $\mathcal{A}$ and every $\mathcal{S}$. The discarding decisions throughout the first part of the process are made according to a series of geometric parameters that are invariant under rigid motion. These parameters help us to describe and compare the shapes of $\mathcal{A}$ and the different subsets of $\mathcal{B}$ that we explore. To navigate $\mathcal{B}$ and have easy access to its subsets, we use a *compressed octree* [9]. By doing this we achieve a reduction of the total computational time, corresponding to a pruning of the search space, as an effect of all the calculations we avoid by discarding parts of $\mathcal{B}$ cheaply and at an early stage.



Overview of all the algorithms used:

LFA: Losless Filtering Algorithm,

OBA: Octree Building Algorithm

COBA: Compressed Octree Building Algorithm

AGIA: Adding Geometric Information Algorithm

SA: Search Algorithm

CPSMA: Colored Point Set Matching Algorithm

E: Enumeration,

T: Testing

The candidate zone determination algorithm consists itself on two subparts. An octree construction algorithm and a search algorithm that traverses the octree looking for the candidate zones. The octree construction algorithm can also be subdivided in two more parts: a compressed octree building algorithm that uses the colored points in $\mathcal{B}$ as sites (without considering their color), and an algorithm that adds the information related to the geometric parameters being used to each node (See also Figure 4).The **NCPSM** algorithm works as described previously, notice that the compressed octree that we build in the first part helps us to construct the octrees needed in the testing algorithm.

4.1 Octree construction

First we build a compressed octree $\mathcal{O}_{\mathcal{B}}$ using the (colored) points as sites. We use the techniques in [9] to ensure a total asymptotic cost of $O(m \log m)$ in all cases, where $m = |\mathcal{B}|$. Then, we add the geometric information considered to $\mathcal{O}_{\mathcal{B}}$. To simplify explanations we consider $\mathcal{O}_{\mathcal{B}}$ to be complete. Although it is clear that this is not the general situation this limitation can be easily overcome in all the parts of the algorithm.

At this stage $\mathcal{O}_{\mathcal{B}}$ contains no information about the different colors of the points in $\mathcal{B}$ or the geometric characteristics of $\mathcal{B}$ as a whole. Since these parameters will guide our search for matches they must be invariant under rigid motion. The geometric parameters we use are: a) parameters that take into account the fact that we are working with point sets: number of points and histogram of points' color attached to a node; b) parameters based on distances between points: maximum and minimum distance between points of every different color. For every geometric parameter we will define a *parameter compatibility criterium* that will allow us to discard zones of the plane that cannot contain a subset $\mathcal{B}'$ of $\mathcal{B}$ to which $\mathcal{A}$ may match. An important asset of our algorithm that enhances its applicability is that other general geometric parameters may be considered in future work as well as specific ones due to specialized applications of the algorithms.

Once selected the set of geometric parameters to be used, in the second stage of the octree construction, we traverse $\mathcal{O}_{\mathcal{B}}$ and associate to each node the selected geometric parameters. We also compute them for the whole of $\mathcal{A}$. The computational cost of adding the geometric information to $\mathcal{O}_{\mathcal{B}}$ depends on the parameters that we choose. In the case of the "number of points" and "histogram of points' colors" parameters we can easily keep track of them while we build the octree, so no additional cost is needed. For the "minimum and maximum distance between points of the same color" parameters, the necessary calculations can be carried out in $O(m^2 \log m)$ time for each color category.

4.2 Lossless Filtering algorithm

The subdivision of $\mathbb{R}^3$ induced by a certain level of the Octree is formed by axis-parallel cubes. To take advantage of this, in the filtering step we search for a certain axis-parallel cube in the octree big enough to contain set $\mathcal{A}$ *even if it appears rotated.* We also demand the cube that we are looking for to contain a part of $\mathcal{B}$ similar to $\mathcal{A}$ in terms of some the geometric parameters described in section 4.1. By doing this, we are able to temporarily forget about all the possible motions that set $\mathcal{A}$ may undergo and just find those zones of the octree where they may actually appear. This type of search is much more adequate to the octree data structure.

Through the rest of the paper, all tetrahedrons and cubes considered are axis-parallel unless explicitly stated. Let $T_{\mathcal{A}}$ be the minimal tetrahedron that contains all the (colored) points in $\mathcal{A}$, and let s be the smallest positive integer for which (diagonal($T_{\mathcal{A}}$) + 2ϵ) $\leq 2^s$ holds.Let us also denote any cube with side length 2^s as a square of *size s*. Note that we use powers of two as side lengths of the cubes considered to simplify the explanations in section 4.2, although it is not really necessary for our algorithm. For any rigid motion τ there exists a cube of side s containing all the points in $\tau(\mathcal{A})$. This allows us to affirm that, for any $\mathcal{S} \subset \mathcal{B}$ that matches $\mathcal{A}$, there exists a cube of side s that contains it. We store the points in $\mathcal{B}$ in a compressed octree $\mathcal{O}_{\mathcal{B}}$ and describe the geometry of each of the nodes in this octree by using the rigid-motion-invariant geometric parameters described in 4.1. Then we look for candidate zones in the octree whose associated geometric parameters match those of $\mathcal{A}$.

To sum up, we can say that, in the first step of the algorithm, instead of looking for all possible rigid motions of set $\mathcal{A}$, we look for *c*ubes of side s covering subsets of $\mathcal{B}$, which are parameter compatible with $\mathcal{A}$. Notice that, although our intention would be to describe our candidate zones exactly as cubes of size s this will not always be possible, so we will also have to use groups of two, four or eight cubes of size s. It is important to stress the fact that ours is a conservative algorithm, so we do not so much look for candidate zones as rule out those regions where no candidate zones may appear. The subdivision induced by the nodes of size s of $\mathcal{O}_{\mathcal{B}}$ corresponds to a grid of cubes of side s superimposed to set $\mathcal{B}$. There are only four ways to place the cube we are looking for respect to this grid that correspond to the relative position of one of the cube's vertices (inside a cube, inside a face, in an edge or on a vertex). This yields four different kinds of candidate zones associated to one, two, four or eight nodes. The subsets $\mathcal{B}'$ that we are looking for may lie anywhere inside those zones.

4.2.1 Search algorithm

We provide a brief overview of an algorithm that traverses the octree $\mathcal{O}_{\mathcal{B}}$ searching for the collection $\mathcal{C}$ of candidate zones. Algorithm 1 presents its main guidelines.

The hierarchical decomposition of $\mathcal{B}$ provided by $\mathcal{O}_{\mathcal{B}}$ makes it possible to begin searching at the whole of $\mathcal{B}$ and later continue the search only in those zones where, according to the selected geometric parameters, it is really necessary. The algorithm searches recursively in all the octants considering also those zones that can be built using parts of more than one of them. The zones taken into account through all the search are easily described in terms of $\mathcal{O}_{\mathcal{B}}$'s nodes and continue to decrease their size, until they reach s, following the algorithm's descent of the octree. Consequently, early discards made on behalf of the geometric parameters rule out of the search bigger subsets of $\mathcal{B}$ than later ones.

Given that two, four or eight nodes defining a candidate zone need not be in the same branch of $\mathcal{O}_{\mathcal{B}}$, at some points we will need to be exploring two, four or eight branches simultaneously. This will force us to have four separate search functions, depending on the type of candidate zones we are looking for, and to keep geometric information associated to those zones that do not correspond exactly to single nodes in the octree but to couples, quartets or octets. The main search function, denoted search_1, seeks for candidate zones formed by only one node and invokes itself and the other three search functions, called search_2, search_4 and search_8 respectively. Consequently, search_2 finds zones formed by pairs of nodes and also launches itself, search_4 and search_8. Analogously, search_4 finds zones formed by quartets of nodes and calls itself and search_8. Finally, search_8 locates zones formed by groups of eight nodes and only invokes itself.

The search step begins with a call to function search_1 with the root node as the parameter. We denote t the size of the root and assume $t \geq s$. Function search_1 begins testing if the information in the current node is compatible to the information in $\mathcal{A}$. If this doesn't happen, there is no possible matching contained entirely in the descendants of the current node and we have finished. Otherwise, if the current node has size s then we have found a candidate zone. If this does not happen, we must go down a level on the octree. To do so, we consider the eight sons of the current node $(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)$.

The candidate zones can be located: **Inside any of the s_i.** So we have to call search_1 recursively in all the s_i's. **Partially overlapping two of the s_i's.** In this case, we would need a function to search both subtrees for all possible pairs of nodes (or quartets) that may arise below in the subdivision. This is function search_2. **Partially overlapping four s_i's.** In this case, we would invoke the function that traverses all four subtrees at a time, called search_4. **Partially overlapping all of the eight s_i's.** in this case we would call function search_8.

Algorithm 1 Search_1(node N)

```
for all S sons of N do
  if (S is parameter compatible with $\mathcal{A}$ ) then
    if ( We have not reached the node size to stop the search) then
      Call Search_1(S)
    else {We have found a candidate node}
      Report candidate zone
    end if
  end if
end for
{Continue in pairs of nodes if necessary (twelve possibilities)}
for all $S_1, S_2$ pairs of neighboring sons of N do
  if (The couple $(S_1, S_2)$ is parameter compatible with $\mathcal{A}$) then
    if ( We have not reached the node size to stop the search) then
      Call Search_2($S_1, S_2$)
    else {We have found a candidate pair}
      Report candidate zone
    end if
  end if
end for
{Continue in quartets of nodes if necessary (four possibilities) }
for all $(S_1, S_2, S_3, S_4)$: Quartets of neighboring sons of N. do
  if (($S_1, S_2, S_3, S_4$) are parameter compatible with $\mathcal{A}$ ) then
    if ( We have not reached the node size to stop the search) then
      Call Search_4 ($S_1, S_2, S_3, S_4$)
    else {We have found a candidate quartet}
      Report candidate zone
    end if
  end if
end for
{Continue in the octet of nodes formed by all the eight sons}
$(S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8)$: Octet formed by the sons of N.
if (($S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8$) are parameter compatible with $\mathcal{A}$ ) then
  if ( We have not reached the node size to stop the search) then
    Call Search_8 (($S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8$))
  else {We have found a candidate octet}
    Report candidate zone
  end if
end if
```

Functions search_2, search_4 and search_8 work similarly but take into account that they need two, four and eight parameters respectively that those must be chosen adequately. The process goes on recursively until the algorithm reaches the desired size (s), yielding a set of candidate zones of all four possible types.

Lemma 4.1. *The number of candidate zones $c = |\mathcal{C}|$ is $O(\frac{m}{n})$. This bound is tight.*

Proof. Each point in $\mathcal{B}$ belongs to a unique node of $\mathcal{O}_{\mathcal{B}}$, it can be seen that each node may belong to up to 27 zones (one of type one, six of type two, twelve of type four and eight of type eight) and thus each point in in $\mathcal{B}$ may belong to, at most, 27 candidate zones. Subsequently, $c \in O(m)$. To improve this bound we consider n_i, the number of points inside the ith candidate zone. As each colored point belongs to at most 27 zones, $\sum_{c_i \in \mathcal{C}} n_i \leq 27m$. As every candidate zone must contain, at least, n points then $cn \leq \sum_{c_i \in \mathcal{C}} n_i$, putting this two statements together, we obtain $c \leq \frac{27m}{n}$. The tightness of the bounds follows from considering, for example, the case when $\mathcal{A} = \mathcal{B}$. $\square$

Lemma 4.2. *a)The total cost of the Search algorithm is $O(m)$. b) The total cost of the lossless filtering algorithm is $O(m^2 \log m)$.*

Proof. a) Through the search algorithm every node is traversed at most 27 times corresponding to the different candidate zones it may belong to, as the compressed octree data strucure guarantees that there are at most $O(m)$ nodes the total computational cost is $O(m)$. b) The result follows from considering the sepparate (additive) contributions of the $O(m^2 \log m)$ Octree building algorithm and the $O(m)$ contribution of the search algorithm. $\square$

Lemma 4.3. *The computational cost of the algorithm that combines the lossless filtering algorithm and the matching algorithm is $O(n^6 m^5 \lambda_q(nm) t(n, n'))$. The bound is tight.*

Proof. The lossless filtering step takes $O(m^2 \log m)$ computational time. Given the $O(n^6 m^5 \lambda_q(nn') t(n, n'))$ cost for every candidate zone, $\mathcal{S}$ with $n' = |\mathcal{S}|$, when all candidate zones are considered, the total cost is $\sum_{\mathcal{C}_i \in \mathcal{C}} O(n^6 {n_i}^5 \lambda_q(nn_i) t(n, n_i))$ where $n_i = |\mathcal{C}_i|$. Bearing in mind that $n_i \leq m$ and factorizing we obtain $T \in \lambda_q(nm) n^6 t(n, n') \sum_{\mathcal{C}_i \in \mathcal{C}} O({n_i}^5)$, taking into account that $\sum_{\mathcal{C}_i \in \mathcal{C}} O({n_i}^5) \leq (\sum_{\mathcal{C}_i \in \mathcal{C}} O(n_i))^5$, $T \in \lambda_q(nm) n^6 t(n, n') (\sum_{\mathcal{S} \in \mathcal{C}} O(n_i))^5$, as each point belongs to at most 27 candidate zones then $\sum_{\mathcal{C}_i \in \mathcal{C}} O(n_i)$ is in $O(m)$ and thus, the result follows. The tightness of the bound is reached, for example, when $\mathcal{A} = \mathcal{B}$ □

5 Conclusions and Future work

The lemma that we have just presented shows that from a formal point of view, our process takes, at its worst, the same computational time as the algorithm that does not use the lossless filtering step. Consequently we benefit from any reduction of the computational time that the filtering achieves without any increase in the asymptotic costs. In practice we expect the assumptions presented in section 3.2 to be met and the $\lambda_q(nm)$ factor to be to be linear in nm and, thus, obtain a $O(n^7 m^6 \log n)$ algorithm that improves slightly the best computational costs for the problem up to date. Given the high computational complexity of the problem, one of the aims in our future work is the development of an implementable approximate algorithm that reduces the computational cost of the exact algorithm presented in this paper at the cost of some of its precision. This type of approximation does not refer to the fuzziness of data, but to how much does the solution given by the algorithm resemble the optimal (taking the noise into account) solution.To do so we intend to adapt some of the ideas in [10] to the 3D case. Finally, we also intend to develop more of the geometric parameters used in the lossless filtering step taking advantage of the geometric properties of 3D space.

References

[1] C. Ambühl, S. Chakraborty, and B. Gartner. Computing largest common point sets under approximate congruence. *I*n Proc. 8th Annual European Symposium on Algorithms, LNCS 1879, pages 52-63, 2000.

[2] H. Alt, K. Mehlhorn, H. Wagener and E. Welzl. Congruence, similarity and symmetries of geometric objects. *Discrete & Computational Geometry*, 3:237–256, 1988.

[3] S. Biswas, S. Chakraborty, Fast Algorithms for Determining Protein Structure Similarity, *W*orkshop on Bioinformatics and Computational Biology, at the International Conference on High Performance Computing (HiPC), Hyderabad, India December 2001.

[4] B. Chazelle, H. Edelsbrunner, L.J. Guibas and M. Sharir, "A singly-exponential stratification scheme for real semi-algebraic varieties and its applications", *Proc. 16th Internat. Colloq. Automata Lang. Program., Lecture Notes Comput. Sci.* Vol. 372, Springer-Verlag, pp 179-192, 1989.

[5] V. Choi, N. Goyal, A Combinatorial Shape Matching Algorithm for Rigid Protein Docking. *CPM 2004, LNCS 3109, pp. 285-296, 2004.*

[6] Y.Diez, J.A. Sellarès. Noisy Disk Set Matching Under Rigid Motion. *Proceedings EWGC*, 115-118, 2006.

[7] F. Dupuis, J. Sadoc, J. Mornon, Protein Secundary Structure Assignment through Voronoi Tesellation *PROTEINS: Structure, Function and Bioinformatics 55:519-528(2004).*

[8] A. Efrat, A. Itai and M.J. Katz. Geometry helps in Bottleneck Matching and related problems. *Algorithmica*, 31:1–28, 2001.

[9] D. Eppstein, M.T. Goodrich, and J.Z. Sun. The skip quadtree: a simple dynamic data structure for multidimensional data. *21st ACM Symp. on Comp. Geom.*, 296–305, 2005.

[10] P.J. Heffernan S. Schirra. Approximate decision algorithms for point set congruence. *Computational Geometry: Theory and Applications* 4(3), 137–156, 1994.

[11] J.E. Hopcroft and R.M. Karp, An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *S*IAM Journal on Computing, 2(4):225–231, Dec. 1973.

[12] M. Sharir, "Recent Developments in Theory of Arrangements of Surfaces", *Lecture Notes in Computer Science*, Vol 1738, pp 1-21, Springer-Verlag 2000.

[13] M. Sharir and P.K. Agarwal, Davenport-Schinzel sequences and their geometric aplications, Cambridge University Press, 1995.