Noisy disk set matching under rigid motion

Yago Diez and J. Antoni Sellarès*

Abstract

Let $\mathcal{A}$ and $\mathcal{B}$ be two disk sets, with $|\mathcal{A}| \leq |\mathcal{B}|$. We propose a process for determining matches between $\mathcal{A}$ and subsets of $\mathcal{B}$ under rigid motion, assuming that the position of all disks in both sets contains a certain amount of "noise". The process consists on two main stages: a candidate zone determination algorithm and a matching algorithm. A candidate zone is a region determined by one, two or four squares that contains 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 quadtree to have easy access to the subsets of $\mathcal{B}$ related to candidate zones. In each quadtree node we store geometric information that is used by the algorithm that searches for candidate zones. The second algorithm solves the disk set matching problem: we generate all, up to a certain equivalence, possible motions that bring $\mathcal{A}$ close to some subset $\mathcal{B}'$ of every $\mathcal{S}$ and seek for a matching between sets $\mathcal{A}$ and $\mathcal{B}'$.

1 Introduction

Determination of the presence of a geometric pattern in a large set of objects is a fundamental problem in computational geometry. It arises in diverse applications such as astronomy (constellation recognition problem) and molecular biology (substructure search problem). Stars can be seen as disks in $\mathbb{R}^2$ with radii determined by their brightness and an atom in a protein molecule can be modelled as a ball in $\mathbb{R}^3$ whose radius is the Van Der Waals radius of the element it represents. Since star and atom positions are fuzzy, both problems can be transformed to approximate disk/ball set matching under rigid motion problems. In this paper we will concentrate in the two-dimensional case.

1.1 Problem formulation

A *rigid motion* in $\mathbb{R}^2$ is a distance preserving mapping that can be expressed as a composition of a rotation and a translation. Fixed a real number $\epsilon \geq 0$ we say that two disks $D(a,r)$, $D(b,s)$ of centers a, b and radii r, s *approximately match* when $r = s$ and $d(a,b) \leq \epsilon$, where d denotes the Euclidean distance.

Let $\mathcal{D}$, $\mathcal{S}$ be two disk sets of the same cardinality. A *radius preserving bijective mapping* $f : \mathcal{D} \to \mathcal{S}$ maps each disk $A = D(a,r) \in \mathcal{D}$ to a distinct and unique disk $f(A) = D(b,s) \in \mathcal{S}$ so that $f(a) = b$ and $r = s$. Let $\mathcal{F}$ be the set of all radius preserving bijective mappings between $\mathcal{D}$ and $\mathcal{S}$. Observe that when the disks in $\mathcal{D}$ and $\mathcal{S}$ have a high number of different radii, $|\mathcal{F}|$ may severely diminish. The *bottleneck distance* between $\mathcal{D}$ and $\mathcal{S}$ is defined as:

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

The **Noisy Disk Matching (NDM)** problem can be formulated as follows. Given two disk sets $\mathcal{A}$, $\mathcal{B}$, $|\mathcal{A}| \leq |\mathcal{B}|$, and $\epsilon \geq 0$, determine all rigid motions τ for which there exists a subset $\mathcal{B}'$ of $\mathcal{B}$ such that $d_b(\tau(\mathcal{A}), \mathcal{B}') \leq \epsilon$.

If τ is a solution to the **NDM** problem, every disk of $\tau(\mathcal{A})$ approximately matches to a distinct and unique disk of $\mathcal{B}'$ of the same radius, and we say that $\mathcal{A}$ and the subset $\mathcal{B}'$ of $\mathcal{S}$ are *noisy congruent*. According to our initial motivation, we are only interested in subsets $\mathcal{B}'$ so that $\mathcal{A}$ and $\mathcal{B}'$ are noisy congruent, and not so much in the individual matchings between disks in $\tau(\mathcal{A})$ and $\mathcal{B}'$.

If we think of a point as a disk of 0 radius and point sets of the same cardinality are considered, then the **NDM** problem becomes the **Noisy Matching (NM)** problem: Given two point sets $\mathcal{A}$, $\mathcal{B}$ of the same cardinality and $\epsilon \geq 0$, determine, if possible, a rigid motion τ such that $d_b(\tau(\mathcal{A}), \mathcal{B}) \leq \epsilon$.

1.2 Previous Results

The study of the **NM** problem was initiated by Alt *et al.* [1] who presented an exact $O(n^8)$ time algorithm for solving the problem for two sets of cardinality n. This bound can be reduced to $O(n^3 \log n)$ if an assignment of points in $\mathcal{A}$ to points in $\mathcal{B}$ is given [1]. Combining Alt et alt. algorithm with the techniques by Efrat *et al.* [3] the time can be reduced to $O(n^7 \log n)$.

We must note here that none of these approaches considers the possibility of working with disk sets and that the fact of having sets of different cardinality is often not considered.

2 Our Approach

Our main goal is to discretize the **NMD** problem by turning it into a series of "smaller" instances, expecting that their solution will be faster. To do so, we will use a conservative strategy to discard those subsets of $\mathcal{B}$ where no noisy match may happen and keep a number of zones where this matches may occur.

*Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain, {ydiez,sellares}@ima.udg.es. Partially supported by grant TIN2004-08065-C02-02.

We will assume that all rectangles and squares we consider are axis-parallel. Our algorithm consists on two main parts. The first one yields a collection of *candidate zones*, which are regions determined by one, two or four squares that contain a subset $\mathcal{S}$ of $\mathcal{B}$ such that $\mathcal{A}$ may approximately match one or more subsets $\mathcal{B}'$ of $\mathcal{S}$. The second part of the algorithm solves the **NDM** problem between $\mathcal{A}$ and every $\mathcal{B}'$.

The discarding decisions throughout the first part of this process will be made according to a series of geometric parameters, invariant under rigid motion, that will help us to describe the shapes of $\mathcal{A}$ and the different subsets of $\mathcal{B}$ that we explore. To navigate $\mathcal{B}$ and have easy access to those subsets, we will use a *compressed quadtree* [2]. By doing this we intend to 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.

The candidate zone determination algorithm consists itself of two subparts: a quadtree construction algorithm and a search algorithm that traverses the quadtree looking for the candidate zones. The quadtree construction algorithm can also be subdivided in two more parts: a compressed quadtree building algorithm that uses the centers of the disks in $\mathcal{B}$ as sites (without considering their radii), and then an algorithm that adds the information related to the geometric parameters being used to each node.

The second part of the algorithm consists on two more parts. The first one, the "enumeration" part, will group all possible rigid motions of $\mathcal{A}$ in equivalence classes in order to make their handling feasible. We will choose a representative motion τ for every equivalence class. The second step, the "testing" part, will perform a bipartite matching algorithm between every set $\tau(\mathcal{A})$ and every disk set $\mathcal{B}'$ associated to a candidate zone. For these matching tests we will modify the algorithm proposed in [3] by using the *skip-quadtree* data structure [4] in order to make it easier to implement and to take advantage of the data structures that we have already built.

3 Candidate zone determination algorithm

Let $R_{\mathcal{A}}$ be the minimal rectangle that contains all the centers of the disks in $\mathcal{A}$, and let s be the smallest positive integer for which $(\text{diagonal}(R_{\mathcal{A}})+2\epsilon) \leq 2^s$ holds. It is not difficult to prove that for any rigid motion τ there exists an square of *size* s (with side length 2^s) containing all the centers in $\tau(\mathcal{A})$. This allows us to affirm that, for any $\mathcal{S} \subset \mathcal{B}$ noisy congruent with $\mathcal{A}$ there will exist a square of size s that contains the centers of its disks. In this first step of our algorithm, instead of looking for all possible rigid motions of set $\mathcal{A}$ we will look for such squares. More specifically, we will store the centers of the disks in $\mathcal{B}$ in a compressed PR quadtree $\mathcal{Q}_{\mathcal{B}}$ and describe the geometry of each of the nodes in this quadtree using a number of geometric parameters that are invariant for rigid motions. Then we will look for candidate zones in the quadtree whose associated geometric parameters match those of $\mathcal{A}$. Although our intention would be to describe our candidate zones exactly as squares of size s this will not always be possible, so we will also have to use two or four squares of size s. It is important to stress the fact that ours is a conservative algorithm, so we will not so much look for candidate zones as rule out those regions where no candidate zones may appear.

3.1 Compressed quadtree construction

Although for the first part of the algorithm we will only use the quadtree levels between the root and the one whose associated nodes have size s, we will use the remaining levels later, so we build the whole compressed quadtree $\mathcal{Q}_{\mathcal{B}}$. We use the techniques in [2] to ensure a total asymptotic cost of $O(m \log m)$ in all cases, where $m = |\mathcal{B}|$.

3.1.1 Adding information to the quadtree

To simplify explanations we will consider $\mathcal{Q}_{\mathcal{B}}$ to be complete. Although it is clear that this will not be the general situation this limitation can be easily overcome in all the parts of the algorithm.

At this moment the quadtree $\mathcal{Q}_{\mathcal{B}}$ contains no information about the different radii of the disks 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. Some examples of geometric parameters we can consider are: a) parameters that take into account the fact that we are working with disk sets: number of disks or list of disk's radii attached to a node; b) parameters based on distances between centers: maximum and minimum distance between centers or maintaining the whole of the distance matrix depending on our practical memory requirements and the performance we achieve. 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 approximately match.

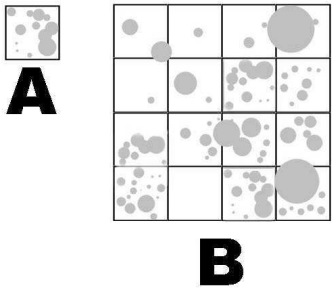

Figure 1: *There cannot be any $\mathcal{B}'$ that approximately matches $\mathcal{A}$ fully contained in the four top-left squares because $\mathcal{A}$ contains twelve disks and the squares only six.*

Once selected the set of geometric parameters to

be used, in the second stage of the quadtree construction, we will traverse $\mathcal{Q}_\mathcal{B}$ and associate to each node the selected geometric parameters. We will also compute them for the whole of $\mathcal{A}$. The computational cost of adding the geometric information to $\mathcal{Q}_\mathcal{B}$ depends on the parameters that we choose. In the case of the "number of disks" and "list of radii" parameters we can easily keep track of them while we build the quadtree, so no additional cost is needed. Adding other parameters will indeed need extra computational time but will also make the discarding of zones more effective. The balance between this two factors will be an important part of our future work.

3.2 Candidate zones determination

We must face the problem of determining all the candidate zones where squares of size s that cover a subset of $\mathcal{B}$ which is parameter compatible with $\mathcal{A}$ can be located. The subdivision induced by the nodes of size s of $\mathcal{Q}_\mathcal{B}$ corresponds to a grid of squares of size s superimposed to set $\mathcal{B}$. If we bear in mind that we are trying to place a certain square in a grid of squares of the same size, it is easy to see that the only three ways to place one of our squares respect to this grid correspond to the relative position of one of the square's vertices. This will yield three different kinds of candidate zones associated to one, two or four nodes (see Figure 2). The subsets $\mathcal{B}'$ that we are looking for may lie anywhere inside those zones.

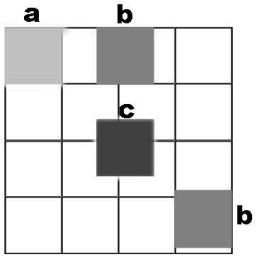

Figure 2: *Position of the candidate zones in the grid.*

3.2.1 Search algorithm

Due to lack of space, we only provide a very brief overview of an algorithm that traverses the quadtree $\mathcal{Q}_\mathcal{B}$ searching for the collection $\mathcal{C}$ of candidate zones. The hierarchical decomposition of $\mathcal{B}$ provided by $\mathcal{Q}_\mathcal{B}$ makes 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 quadrants 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 will continue to decrease their size, until they reach s, following the algorithm's descent of the quadtree. Consequently, early discards made on behalf of the geometric parameters will rule out of the search bigger subsets of $\mathcal{B}$ than later ones.

Given that two or four nodes defining a candidate zone need not be in the same branch of $\mathcal{Q}_\mathcal{B}$, at some points we will need to be exploring two or four branches simultaneously. This will force us to have three separate search functions, depending on the type of candidate zones we are looking for, and to calculate the geometric information associated to those zones that do not correspond exactly to nodes in the quadtree. This calculations are of constant cost in the case of the parameters number of disks and list of radii. The main search function seeks for candidate zones formed by only one node and the other two seek for zones formed by two or four nodes.

In the worst case all possible zones are considered to be candidate zones and the total number of nodes of $\mathcal{Q}_\mathcal{B} \in O(m)$, so the number of candidates zones, $c = |\mathcal{C}|$, is in $O(m)$. Considering a "perfect" behavior of the geometric pruning technique and that the quadtree is descended from the root (sized t) down to a level whose nodes have size s, $t - s < m$, then the cost of the search algorithm is $O(c(t-s)) \in O(m^2)$.

Summarizing, the computational cost of our algorithm in the case where in the two steps only the "number of disks" and "list of radii" parameters are used is $O(\max\{m \log m, c(t-s)\}$. In practice we expect the factor $(t-s)$ to be close to constant achieving computational times close to quasi-linear in m.

4 NDM problem solving algorithm

Now we have a **NDM** problem where we can expect the sets involved, $\mathcal{A}$, $\mathcal{S} \in \mathcal{C}$ ($n = |\mathcal{A}| \leq n' = |\mathcal{S}| \leq m$), to be "similar" in numbers of disks and in the aspects of their shape described by the geometric parameters. We present an algorithm to solve this **NDM** problem, that adapts the best currently existing algorithms for solving the **NM** problem [3, 1] and takes advantage of the compressed quadtree that we have already built and is implementable. Our approach will consist of two parts called "enumeration" and "testing".

4.1 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. We will partition the set of all rigid motions in equivalence classes in order to make their handling possible.

For $b \in \mathbb{R}^2$, let $(b)^\epsilon$ denote the circle of radius ϵ centered at point b. Let $\mathcal{S}^\epsilon$ denote the set $\{(b)^\epsilon | D(b,s) \in \mathcal{S}\}$. Consider the arrangement $\mathcal{A}(\mathcal{S}^\epsilon)$ induced by the circles in $\mathcal{S}^\epsilon$. Two rigid motions τ and τ' will be considered equivalent if for all disks $D(a,r) \in \mathcal{A}$, $\tau(a)$ and $\tau'(a)$ lie in the same cell of $\mathcal{A}(\mathcal{S}^\epsilon)$. We generate a solution in each equivalence class, when it exists, and its corresponding representative motion using the techniques presented in [1]. A simple geometric argument shows that if there exists any rigid motion τ that solves our **NDM** problem then there exists an-

other rigid motion τ', that belongs to the equivalence class of τ, that also does it and such that we can find two pairs of disks $D(a_i, r_i), D(a_j, r_j) \in \mathcal{A}$ and $D(b_k, s_k), D(b_l, s_l) \in \mathcal{S}$, $r_i = s_k$ and $r_j = s_l$, with $\tau'(a_i) \in (b_k)^\epsilon$ and $\tau'(a_j) \in (b_l)^\epsilon$. We check this property for all quadruples i, j, k, l.

Mapping a_i, a_j onto the boundaries of $(b_k)^\epsilon$, $(b_l)^\epsilon$ respectively in general leaves one degree of freedom which is parametrized by the angle $\phi \in [0, 2\pi)$ between the vector $|a_i - b_k|$ and a horizontal line. Considering any other disk $D(a_h, r_h) \in \mathcal{A}$, $h \neq i, j$ for all possible values of ϕ, the center of that disk will trace an algebraic curve of degree 6 σ_{ijklh} so that for every value of ϕ there exists a rigid motion τ_ϕ holding $\tau_\phi(a_i) \in (b_k)^\epsilon$, $\tau_\phi(a_j) \in (b_l)^\epsilon$ and $\tau_\phi(a_h) = \sigma_{ijklh}(\phi)$. For every remaining disk $D(b_p, s_p)$ in $\mathcal{S}$ with $s_p = r_h$, we compute the intersections between $(b_p)^\epsilon$ and $\sigma_{ijklh}(\phi)$ which contains at most 12 points. For parameter ϕ, this yields a maximum of 6 intervals contained in $I = [0, 2\pi[$ where the image of $\tau_\phi(a_h)$ belongs to $(b_p)^\epsilon$. We will call this set $I_{p,h}$ following the notations in [1]). Notice for all the values $\phi \in I_{p,h}$ we may approximately match both disks. We repeat the process for each possible pair $D(a_h, r_h), D(b_p, s_p)$ and consider the sorted endpoints, called *critical events*, of all the intervals $I_{p,h}$. Notice that the number of critical events is $O(nn')$. Subsequently, any $\phi \in [0, 2\pi[$ that is not one of those endpoints belongs to a certain number of $I_{p,h}$'s and ϕ corresponds to a certain rigid motion τ_ϕ that brings the disks in all the pairs $D(a_h, r_h), D(b_p, s_p)$ near enough to be matched. The subdivision of $[0, 2\pi[$ consisting in all the maximal subintervals that do not have any endpoints of any $I_{p,h}$ in their interior stands for the partition of the set of rigid motions that we were looking for.

4.2 Testing

We move parameter ϕ along the resulting subdivision of $[0, 2\pi[$. Every time a critical event is reached, we test the sets $\tau_\phi(\mathcal{A})$ and $\mathcal{S}$ for matching. Whenever the testing part determines a matching of cardinality n we annotate the corresponding τ_ϕ and proceed. Following the techniques presented in [3], in order to update the matching, we need to find a single augmenting path using a layered graph.

When searching for augmenting paths we will need to perform efficiently the two following 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**$(D(\mathcal{T}), s)$: Delete point s from $D(\mathcal{T})$. For our implementation we will use the *skip quadtree*, a data structure that combines the best features of a quadtree and a skip list [4]. The cost of building a skip quadtree for any set $\mathcal{T} \subseteq \mathcal{S}$ is in $O(n' \log n')$. This computational cost is, in the worst case when $n' = m$, the same obtained in [3]. **Neighbor** operation is used to get all the points in our skip quadtree holding the condition previously stated, and combined with the delete operation to prevent refinding points. This corresponds to a range searching operation in the skip quadtree followed by a set of deletions. The asymptotic computational cost of the **deletion** of a point in a skip quadtree is $O(\log n')$. The range searching can be approximated in $O(\delta^{-1} \log n' + u)$ time where u is the size of the output in the approximate range query case, for constant $\delta > 0$ [4]. The approximate range searching outputs some "false" neighbor points that can be detected in $O(1)$ time. For the asymptotic cost of the neighbor operation, we observe that the smaller δ is, the bigger the computational time for the approximate range searching but the smaller the number of false neighbors. This leaves the cost of the **neighbor** operation between $O(\frac{\delta^{-1} \log n'}{u} + 1)$ amortized cost, when δ allows no false neighbors, and $O(n')$ when δ is arbitrarily big. The first case meets the cost in [3] and the second one is a little worse and meets the costs in [1]. We believe this last case to be unrealistic and expect an overall of our algorithm performance similar to [3] using simpler data structures. We denote $t(n')$ an upper bound on the time of performing **neighbor** operation in $\mathcal{S}$'s skip quadtree. This yields a computational cost of $O(nt(n'))$ for finding an augmenting path.

4.3 Computational Cost

During the enumeration part, in the worst case $O(n^2 n'^2)$ quadruples of disks are considered. For each quadruple, we work with $O(nn')$ pairs of disks, obtaining $O(nn')$ critical events. Summed over all quadruples the total number of critical events encountered in the course of the algorithm is $O(n^3 n'^3)$. Finally for the testing part, since we spent $O(nt(n'))$ time at each critical event for finding an augmenting path, the total time of the algorithm sums to $O(n^4 n'^3 t(n'))$.

When all candidate zones $\mathcal{S} \in \mathcal{C}$ are considered, the total cost is $\sum_{\mathcal{S} \in \mathcal{C}} O(n^4 n'^3 t(n'))$.

References

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

[2] S. Aluru and F.E. Sevilgen. Dynamic compressed hyperoctrees with application to the N-body problem. *Proc. 19th Conf. Found. Softw. Tech. Theoret. Comput. Sci.*, LNCS 1738:21-33, 1999.

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

[4] 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.