Doctoral Dissertation

Geometric Optimization Problem Solving: Matching Sets of Line Segments and Multi-robot Path Planning

Hyeyun Yang

Department of Computer Science and Engineering

Ulsan National Institute of Science and Technology

2022

Geometric Optimization Problem Solving: Matching Sets of Line Segments and Multi-robot Path Planning

Hyeyun Yang

Department of Computer Science and Engineering

Ulsan National Institute of Science and Technology

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Geometric Optimization Problem Solving: Matching Sets of Line Segments and Multi-robot Path Planning

A dissertation submitted to
Ulsan National Institute of Science and Technology
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Hyeyun Yang

07.14.2022 of submission

Approved by

[illegible]

Advisor

Antoine Vigneron

Geometric Optimization Problem Solving: Matching Sets of Line Segments and Multi-robot Path Planning

Hyeyun Yang

This certifies that the dissertation of Hyeyun Yang is approved.

07.14.2022 of submission

Signature

[illegible]

Advisor: Antoine Vigneron

Signature

H. K. Ahn

Hee-Kap Ahn

Signature

Sang Won Bae

Sang Won Bae

Signature

[illegible]

Tsz-Chiu Au

Signature

Ilwoo Lyu

Ilwoo Lyu

Abstract

We study two geometric optimization problems: Line segments pattern matching and multi-robot path planning. We give approximation algorithms for matching two sets of line segments in constant dimension. We consider several versions of the problem: Hausdorff distance, bottleneck distance and largest common subset. We study these similarity measures under several sets of transformations: translations in arbitrary dimension, rotations about a fixed point and rigid motions in two dimensions. As opposed to previous theoretical work on this problem, we match segments individually, in other words we regard our two input sets as *sets* of segments rather than *unions* of segments.

Then we consider a multi-robot path planning problem. A collection of square robots need to move on the integer grid, from their given starting points to their target points, and without collision between robots, or between robots and a set of input obstacles. We designed and implemented three algorithms for this problem. First, we computed a feasible solution by placing middle-points outside of the minimum bounding box of the starting positions, the target positions and the obstacles, and moving each robot from its starting point to its target point through a middle-point. Second, we applied a simple local search approach where we repeatedly delete and insert again a random robot through an optimal path. It improves the quality of the solution, as the robots no longer need to go through the middle-points. Finally, we used simulated annealing to further improve this feasible solution.

Contents

List of Figures

List of Tables

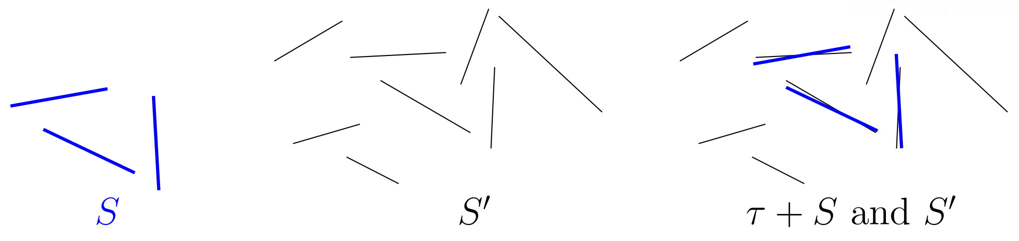


Figure 1: Matching two sets of segments under translation.

I Introduction

Computational geometry is a field of computer science that deals with geometric objects such as points, lines, polygons, polytopes .... In this thesis, we study two computational geometry problems, and more precisely, *geometric optimization* problems. Geometric optimization consists in minimizing or maximizing a cost function defined for a set of such geometric objects.

Ideally, we would want to find an algorithm that (1) returns an optimal solution, (2) is fast, and (3) solves any instance of the problem. In complexity theory, it is often desirable to find such a solution in polynomial time. If we cannot find such an algorithm, we may relax some of the conditions (1), (2) and/or (3). For instance, if we relax Condition (1) and we only return a solution whose cost is a constant factor from optimal, still in polynomial time, then we have obtained an *approximation algorithm*. If the solution is not known to be within a constant factor from optimal, we have a *heuristic*, which may give a good solution for most instances, but without any guarantee.

Geometric pattern matching relates to measuring the similarity between geometric objects such as polygons or sets of points. These problems are motivated by applications to computer vision, for instance. When we want to find an optimal matching, it becomes a geometric optimization problem. Path planning relates to computing a path for a robot, or a set of robots, between a starting configuration and a target configuration. If we want to find a path that is optimal according to some criterion, it also becomes a geometric optimization problem.

In this thesis, we studied pattern matching problems where the input geometric objects are line segments. An image can be represented in different ways. For instance, it can be represented by a bitmap, which is a matrix of pixels. A different approach is to give a geometric representation of the image as a set of line segments. The line segments could represent, for instance, the edges of a building or a corridor. Such a representation may save space as an edge is represented by only its two endpoints, and it also gives more structured information than a raw bitmap.

Path planning for a single robot has been studied a lot. Nowadays, multi-robot path planning is used in many industrial applications. For instance, unmanned vehicles are used in warehouses to move objects quickly without collision. These robots work in 2D Euclidean space, and move continuously. There are warehouses arranged as grids like Autostore and Ocado. Each bin on the grid is shifted by robots moving on the top of the grid and is sent to a portal where human

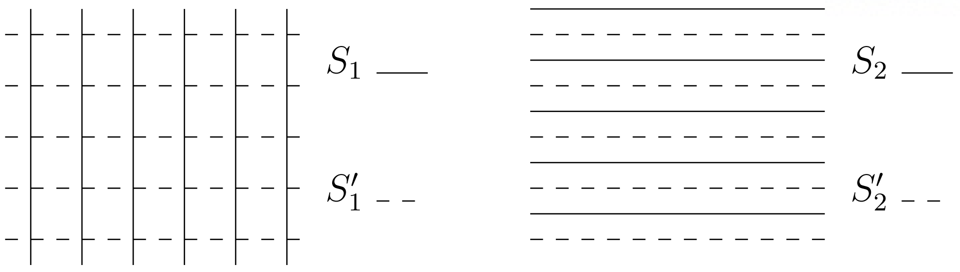


Figure 2: On the left, the *union* of the solid segments $\bigcup S_1$ is similar to the union of the dashed segments $\bigcup S_1'$ in the sense any point on a dashed segment is close to a point on a solid segment. However, taken individually, no segment in $s \in S_1$ is similar to any segment $s' \in S_1'$ as they are orthogonal. Our approach ensures that S_1 and S_1' will be considered dissimilar, whereas S_2 and S_2' (right) are similar as the endpoints match closely.

workers wait for the products to be delivered in the bins. What if each bin can move on the grid? Then this problem is similar with the problem we study in this thesis. However, there are still some differences. For example, we considered the 2D grid case, and we have unbounded space for the robots to move.

1.1 Matching sets of line segments

In Section II, we study line segments pattern matching. Line segments pattern matching consists in measuring the similarity between two sets of line segments, or finding a transformation that makes them as similar as possible. (See Figure 1.) Several worst-case efficient algorithms have been designed for this type of problems. However, these algorithms consider *unions* of segments $\bigcup S$ and $\bigcup S'$ instead of *sets* of segments S and S': the goal is to find a matching such that each point on any segment of the first set S is close to a segment in the other set S'. (See the survey by Alt and Guibas [1].) The *union* of segments $\bigcup S$ is $\bigcup S = \{p \mid \exists s \in S : p \in s\}$, and the *set* of segments S is $S = \{s \mid s \in S\}$. Figure 2 shows an example where these two notions of similarity differ substantially. These algorithms give a high similarity between two sets of orthogonal line segments such as the left one in Figure 2, even though their endpoints are far from each other and the angle between two segments from different sets is 90 degree. Based on this observation, we will rather compute the distance between the endpoints of the line segments. As described in the survey [1], the algorithms computing an optimal solution under a set of transformations have a large running time. So we will propose an approximation scheme for matching sets of line segments.

Our goal in this paper is to match sets of line segments, in the sense that two segments are matched if their endpoints are close, and then the sets S_1 and S_1' from Figure 2 would be considered dissimilar, while S_2 and S_2' would be a good match. Our main contribution is to show that, under a certain model, this problem can be solved using approximation algorithms, which are about as efficient as the currently known algorithms for point-set pattern matching (within a

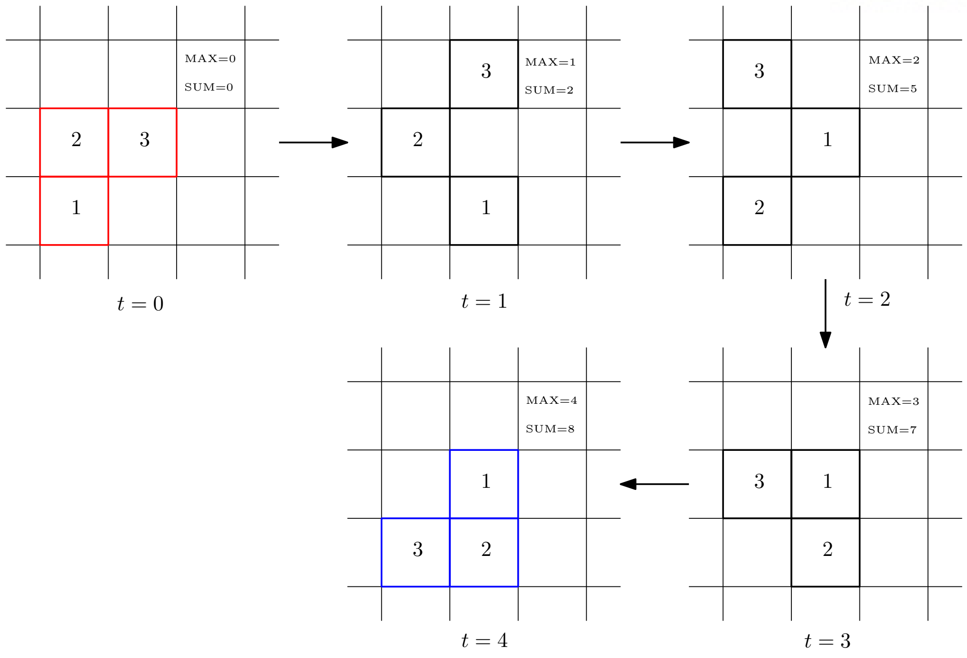


Figure 3: The makespan is 4 and the total number of moves is 8.

10	1	2	3
11	8	9	4
12	7	6	5
15	13	14	(empty pixel)

Figure 4: 15-puzzle problem: We want to place the tiles in numerical order by sliding tiles.

linear factor if we only look at the dependency on the input size).

1.2 Multi-robot path planning

In Section III, we study a multi-robot path planning problem where a set of square robots need to move simultaneously from their given starting positions to their target positions. (See problem description in Section III.) The robots move along a grid, one square at a time. They are not allowed to collide during their movement. Figure 3 shows an example of path planning for three robots. We also allow static obstacles to be placed.

This problem is related to the well-known 15-puzzle (see Figure 4) where tiles numbered 1 to 15 on a 4×4 frame where only one tile is missing, must be moved one by one until they are properly ordered from 1 to 15 when going from left to right and from top to bottom. Our problem is similar, but more general in the sense that we can move several squares simultaneously on the unbounded grid.

We study two versions of the problem: The min-MAX version where we want to minimize the time at which the last robot reaches its target position, and the min-SUM version where we want to minimize the total number of moves of all the robots. The min-MAX version is relevant when we want to complete the motion as soon as possible, and the min-SUM version is relevant if we want to minimize the energy spent.

1.3 Outline

In Section II, our approximate algorithms for matching sets of line segments are described. We address the Hausdorff distance 2.4, bottleneck distance 2.5, and largest common subset 2.6 problems. The static, translation, rotation, and rigid motion cases are also addressed for each problem. The lower bound for matching with respect to the Hausdorff distance is proved in Section 2.8. In Section III, optimization algorithms for multi-robot path planning and experimental results are explained. The comparison between the two heuristic methods is in Section 3.5.2.

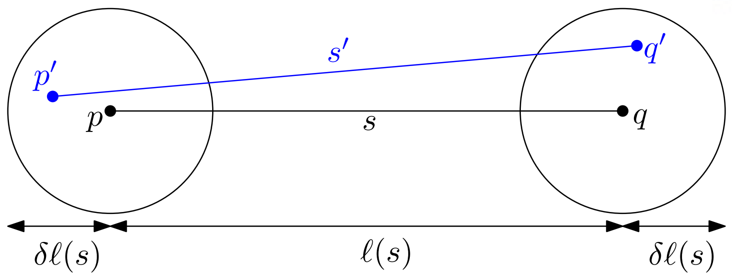


Figure 5: Two segments at distance $d(s, s') \leqslant \delta\ell(s)$.

II Matching sets of line segments

In this section, we present our results on segment pattern matching. Two preliminary versions of this section appeared in the proceedings of the *13th International Conference and Workshops on Algorithms and Computation* [2] and in the *Theoretical Compute Science* [3].

2.1 Problem statements

We are given two sets of line segments $S = \{s_1, \ldots, s_m\}$ and $S' = \{s'_1, \ldots, s'_n\}$ in $\mathbb{R}^d$ such that $m \leqslant n$. Each of these segment s_i or s'_j is directed, and is given by its two endpoints p_i, q_i and p'_i, q'_i respectively. We denote by ℓ_i the length of the segment s_i. The restriction to directed segments is only for ease of presentation; we can handle undirected segments without affecting our time bounds, as explained in Section 2.7.

Our goal is to find a matching between S and S' under a set of transformations $\mathcal{F}$. For instance, when $\mathcal{F}$ is the set $\mathcal{T}$ of translations in $\mathbb{R}^d$, we may want to determine whether there exists a translation $\tau \in \mathcal{T}$ such that $\tau(S) \subset S'$. In practice, however, inaccuracies in the data mean that we cannot hope for an exact match, so we will try to find a translation such that each translated segment of S is close to a segment of S'. (See Figure 1.) We therefore need to introduce a similarity measure for line segments.

2.1.1 Matching Criterion

For any two points $p, q \in \mathbb{R}^d$, we identify the segment $\overline{pq}$ from p to q with the pair of points $(p, q) \in \mathbb{R}^{2d}$. The *distance* $d(s, s')$ between two segments $s = (p, q)$ and $s' = (p', q')$ is the Euclidean distance between these two segments regarded as points in $\mathbb{R}^{2d}$, and thus $d(s, s') = \sqrt{\|p' - p\|^2 + \|q' - q\|^2}$. We say that two segments s and s' match if $d(s, s') \leqslant \delta\ell(s)$, where $\ell(s)$ is the length of the segment s, and $\delta > 0$ is a parameter called *tolerance*.

We use this matching criterion for three main reasons. First, for small values of δ, two matching segments $s = \overline{pq}$ and $s' = \overline{p'q'}$ are similar in the sense that the distance between their endpoints is small, the angle between them is small, and their lengths are approximately the same. (See Figure 5.) More precisely, we have $\|p - p'\| \leqslant \delta\ell(s)$, $\|q - q'\| \leqslant \delta\ell(s)$, the angle between s and s' is $O(\delta)$, and $(1 - \delta\sqrt{2})\ell(s) \leqslant \ell(s') \leqslant (1 + \delta\sqrt{2})\ell(s)$. (Proposition 1.) Conversely,

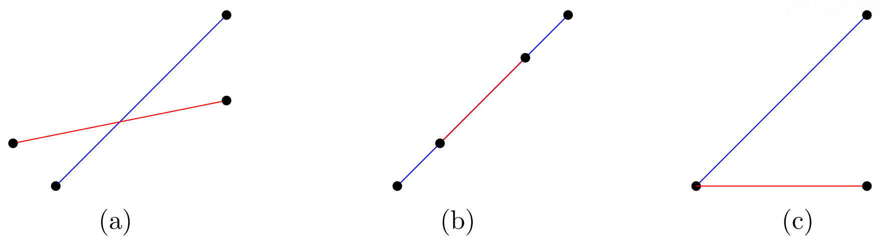


Figure 6: Degenerate cases of line segments.

	Hausdorff	Bottleneck	LCS
Static	$O((m/\varepsilon^{2d}+n)\log n)$	$O((n^{1.5}/\varepsilon^{2d})\log n)$	$O((n^{1.5}/\varepsilon^{2d})\log n)$
Translation	$O((mn/\varepsilon^{3d})\log n)$	$O((n^{2.5}/\varepsilon^{3d})\log n)$	$O((mn^{2.5}/\varepsilon^{3d})\log n)$
2D rotation	$O((mn/\varepsilon^{5})\log n)$	$O((n^{2.5}/\varepsilon^{5})\log n)$	$O((mn^{2.5}/\varepsilon^{5})\log n)$
2D rigid motion	$O((mn^{2}/\varepsilon^{7})\log n)$	$O((n^{3.5}/\varepsilon^{7})\log n)$	$O((m^{2}n^{3.5}/\varepsilon^{7})\log n)$

Table 1: Time bounds of our $(1+\varepsilon)$-approximation algorithms.

if the endpoints are close in the sense that $d(p,p') \leqslant \delta\ell(s)/\sqrt{2}$ and $d(q,q') \leqslant \delta\ell(s)/\sqrt{2}$, then $d(s,s') \leqslant \delta\ell(s)$, and thus the segments match according to our criterion.

The second reason for using this criterion is that it allows us to use approximate near neighbor (ANN) data structures to efficiently compute an approximate nearest segment to a query segment. Otherwise, we need more storage to save the point-set of line segments in $\bigcup S'$. For instance, using the data structure by Arya et al. [4], the query time is $O((1/\varepsilon^{2d})\log n)$ for a segment in $\mathbb{R}^d$ identified with a point in $\mathbb{R}^{2d}$. This will help us design efficient approximation algorithms.

The third reason is that it allows an arbitrary set of line segments. Because we only consider the endpoints of the line segments, the following degenerate cases can be solved: (a) intersecting between line segments, (b) overlapping, and (c) having the same endpoint. (See Figure 6.)

2.1.2 Set of Transformation

We consider different sets $\mathcal{F}$ of transformations. In the *static* case, we do not apply any transformation to our point sets, in other words, we only use the identity transformation. The set $\mathcal{T}$ is the set of *translations* of $\mathbb{R}^d$, so each translation can be represented by a point $\tau \in \mathbb{R}^d$ and it maps any $x \in \mathbb{R}^d$ to $\tau(x) = \tau + x$. We will also consider rotations about a fixed center O in $\mathbb{R}^2$. Finally, we will consider the set $\mathcal{R}$ of rigid motions in $\mathbb{R}^2$, or more precisely, the set of translations and rotations about arbitrary points. We will not consider reflections, as it suffices to run our algorithm on an arbitrary reflected copy of S to cover all possible glide reflections.

2.1.3 Hausdorff distance

We define the *directed Hausdorff distance* $d_H(S,S')$ between S and S' as the minimum value of δ such that, for all $s \in S$, there exists $s' \in S'$ satisfying our matching criterion $d(s,s') \leqslant \delta\ell(s)$.

So it can be expressed as follows: $d_H(S, S') = \max_{s \in S} \min_{s' \in S'} \dfrac{d(s, s')}{\ell(s)}$. In this paper, we do not consider the undirected Hausdorff distance, so we will simply say Hausdorff distance. The Hausdorff distance under the set of transformations $\mathcal{F}$ is the minimum of $d_H(f(S), S')$ over all $f \in \mathcal{F}$. Our approximation algorithms compute a $(1+\varepsilon)$-approximation of this quantity, for some $0 < \varepsilon < 1$. More precisely, we find a transformation $f^\varepsilon \in \mathcal{F}$ such that $d_H(f^\varepsilon(S), S') \leqslant (1+\varepsilon) \min_{f \in \mathcal{F}} d_H(f(S), S')$.

2.1.4 Bottleneck distance

The *bottleneck distance* $d_b(S, S')$ between S and S' is analogous to the Hausdorff distance, except that we require the pairs (s, s') to be matched in a one-to-one manner. So $d_b(S, S') \leqslant \delta$ if there is a one-to-one mapping $\sigma : S \to S'$ such that $d(s, \sigma(s)) \leqslant \delta \ell(s)$ for all $s \in S$.

2.1.5 Largest common subset

The goal is to find the largest subset $C \subset S$ such that there exists a transformation $f \in \mathcal{F}$ that matches C to a subset of S'. So there should be a one-to-one matching between C and a subset of S', such that $d(s, s') \leqslant \delta \ell(s)$ for each matching pair (s, s'). We will relax the problem slightly, and return a matching such that $d(s, s') \leqslant (1+\varepsilon)\delta\ell(s)$ for all matching pair (s, s'), and that has cardinality at least the optimal cardinality for the original problem.

2.1.6 Our results and approach

We obtained $(1+\varepsilon)$-approximation algorithms for all these distance measures under our sets of transformations, when $0 < \varepsilon < 1$. Our algorithms for Hausdorff distance, bottleneck distance and LCS are presented in Section 2.4, 2.5 and 2.6, respectively. In Section 2.7, we briefly explain how to handle undirected line segments. Our results are summarized in Table 1.

Our algorithms first compute a discretization of the set of transformations, and then solve the problem approximately for each transformation in this set using known algorithms: The ANN data structure by Arya et al. [4] in the case of Hausdorff distance, and a geometric matching algorithm by Efrat et al. [5] for the bottleneck distance and the LCS problem.

To be more precise, we first compute a constant-factor approximation of the solution using a coarse discretization. For translations we use the set of vectors $p'_j - p_1$ where $s_1 = (p_1, q_1)$ is assumed to be the shortest segment in S. For rotations about a fixed center O, we choose the angles that align p_a with each point p'_j, where s_a is the segment with largest aspect ratio $\alpha_a = \max(\|p_a\|, \|q_a\|)/\ell_a$.

Then we compute a $(1+\varepsilon)$-approximation by refining these discretizations. In the translation case, we use a uniform grid of $O(1/\varepsilon^d)$ points within a ball of radius proportional to ℓ_1 centered at $p'_j - p_1$. For rotations, we use a set of $(1/\varepsilon)$ equally spaced angles about the angles we chose for obtaining a constant factor approximation, where the spacing is proportional to $1/\alpha_a$.

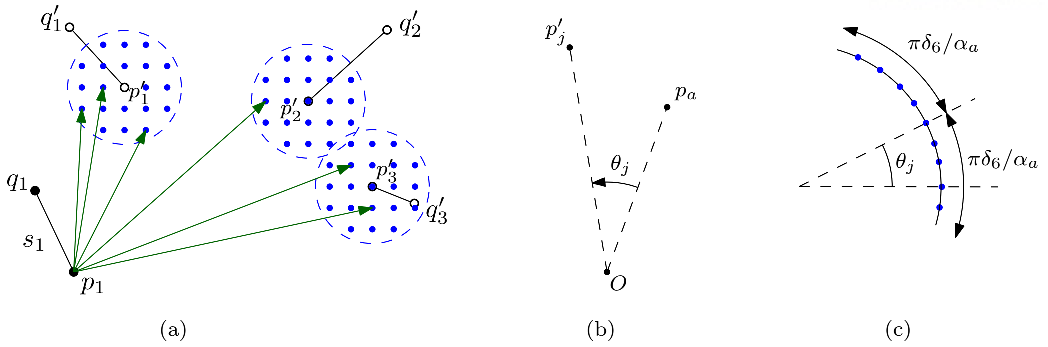


Figure 7: (a) Discretization of the space of translations. (b) The angle θ_j used for a constant factor approximation. (c) Discretization of the set of angles around θ_j for obtaining a $(1+\varepsilon)$-approximation.

(See Figure 7.) For rigid motions, we discretize the space of rigid motion by combining our discretizations for translations and rotations about a fixed center. The main part of our proof is a careful analysis showing that it yields a $(1+\varepsilon)$-approximation of the optimum.

2.2 Related Work

As we mentioned earlier, several algorithms are known for matching line segments, but they consider unions of segments instead of sets of segments. The survey by Alt and Guibas [1] mentions several such algorithms that consider unions of objects, instead of sets of objects [6–8]. These algorithms are therefore adapted for matching polygons, seen as unions of segments and their interior. The algorithm by Alt et al in [6] can compute an asymptotically optimal matching in $O(n \log n)$. Both algorithms by [7,8] use the rectangles 2ε with semidisks of radius ε attached at two endpoints so called "racetracks" [8]. Even though these algorithms apply to arbitrary sets of non-intersecting line segments (so they can handle the intersecting case (c) in Figure6), they still identify the line segments with the sets of points lying on their line segments. These algorithms would not be suitable for the example in Figure 2 and for the set having intersecting line segments. For two sets of axis-parallel line segments with the same cardinality, a matching algorithm with a new criterion called *coverage measure* was designed by Efrat et al [9].

Recent related work presents efficient algorithms for matching polygons or unions of disks, under translations or rigid motions, using the area of overlap as a similarity measure [10–13]. Point-set pattern matching under translation and rigid motion has also been studied extensively. See again the survey by Alt and Guibas [1]. For instance Alt et al. [14] gave exact and approximation algorithms for matching point sets under translations and rigid motions. The translation case was improved by Efrat et al. [5]. Heffernan and Schirra [15] considered decision versions of point set matching under translations and rigid motions. Recently, Yon et al. gave approximation schemes for the largest common subset problem under the same set of transformations [16].

Our approach is based on discretizing the space of translations using grids, and discretizing the angles of rotation uniformly within appropriate intervals. These techniques have been used to obtain some of the results that we already mentioned [10,11,16]. Our technical contribution is to adapt these methods to the problem of matching sets of line segments, which requires the careful analysis presented in Section 2.4. One difference with the case of point-set pattern matching is that our tolerance δ is weighted by the length $\ell(s)$ of the segment s to be matched, hence the segment s_1 with smallest length, and the segment s_a of smallest aspect ratio play a special role in our algorithms and their analysis. Because under the translations, the shortest segment varies more than other segments; under the rotations, the segment with the largest aspect ratio varies more than others. Another issue is that our segments are identified with points in $\mathbb{R}^{2d}$, so exact nearest-neighbor data structures become rather slow, and we need to rely on approximate versions even for dimension $d = 2$.

In terms of running time, we cannot directly compare with previous work because, as far as we know, our problem of matching sets of line segments has not been studied before, and the algorithms for related problems sometimes present extra parameters in the running time [15], or consider different types of objects [11], or are restricted to sets of same cardinality [5,9,15]. So we will only compare the dependency on the input size n, ignoring other parameters. Our algorithms for Hausdorff distance and bottleneck distance under translation run in time $O(n^2)$ and $O(n^{2.5})$, respectively.

The algorithm by Efrat et al. [5] for point sets runs in time $O(n^{1.5}\log n)$, but requires that the sets have same cardinality, which makes the problem easier as it allows to take advantage of corner points. The algorithm by Heffernan et al. [15] for point sets runs in $O(n^{1.5}\log n)$ and has the same restriction. For LCS of point sets under translation, Yon et al. [16] give an $O(n^{3.5}\log n)$ algorithm, as does ours. Our algorithm for bottleneck distance under rigid motion takes time $O(n^{3.5}\log n)$ while Heffernan and Schirra's algorithm achieves $O(n^{2.5}\log n)$ (still for point sets). So overall, our algorithms have a running time that is similar to previous work on point set pattern matching, at most within an $O(n)$ factor, and our algorithms apply to line segments instead of points.

2.3 Preliminary

In this section, we prove two facts mentioned in the introduction. Remember that for any two points $p, q \in \mathbb{R}^d$, we identify the segment from p to q with the pair of points $(p, q) \in \mathbb{R}^{2d}$, and the distance $d(s, s')$ between two segments is the Euclidean distance between these two segments regarded as points in $\mathbb{R}^{2d}$.

Proposition 1. *Suppose that two segments $s = \overline{pq}$ and $s' = \overline{p'q'}$ are matching, which means that $d(s, s') \leqslant \delta\ell(s)$. Then we have:*

(a) $(1 - \delta\sqrt{2})\ell(s) \leqslant \ell(s') \leqslant (1 + \delta\sqrt{2})\ell(s)$.

(b) If $\delta < 1/\sqrt{2}$, then the angle between s and s' is at most $\arcsin(\sqrt{2}\delta)$*, hence it is $O(\delta)$.*

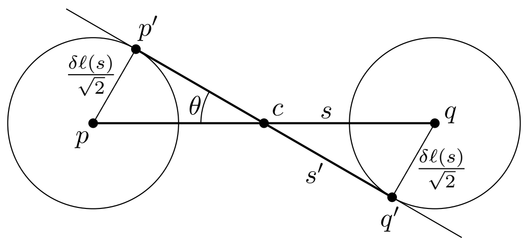

p'
$\frac{\delta\ell(s)}{\sqrt{2}}$
θ
c
s
q
p
$\frac{\delta\ell(s)}{\sqrt{2}}$
s'
q'

Figure 8: Proof of Proposition 1b.

Proof. Suppose that p, q, $\|p - p'\| = \alpha$ and $\|q - q'\| = \beta$ are fixed. Then the distance $\|p' - q'\|$ is maximized when the segment $\overline{pq}$ is contained in $\overline{p'q'}$. So in this case, we have $\ell(s') = \ell(s) + \alpha + \beta$. By our assumptions, $\alpha^2 + \beta^2 \leqslant (\delta\ell(s))^2$, so $\alpha + \beta$ is maximized when $\alpha = \beta = \delta\ell(s)\sqrt{2}/2$. It implies that $\ell(s') = (1 + \delta\sqrt{2})\ell(s)$. This completes the proof of (a). We now prove (b).

We first argue that in the worst case, that is, when the angle θ between s and s' is maximum, then the midpoints c and c' of s and s' coincide. So we start from a configuration where these midpoints coincide, and thus $p - p' = q' - q$. Now suppose we apply a translation $\tau \neq 0$ to s. Then we have

$$\begin{aligned}
&\|\tau + p - p'\|^2 + \|\tau + q - q'\|^2 \\
&= \|p - p' + \tau\|^2 + \|p - p' - \tau\|^2 \\
&= 2\|p - p'\|^2 + 2\tau^2,
\end{aligned}$$

where the second equation follows from Apollonius's theorem.

Therefore, we have $\|\tau + p - p'\|^2 + \|\tau + q - q'\|^2 > \|p - p'\|^2 + \|q - q'\|^2$. It means that $d(\tau + s, s') > d(s, s')$, while the angle between $\tau + s$ and s' is the same as the angle between s and s'. So for a fixed angle θ, the shortest distance $d(s, s')$ is achieved when the midpoints coincide. It implies that, for a given distance $d(s, s')$, the largest angle θ is achieved when their midpoints coincide, because otherwise, we could translate s so that the midpoints coincide, and rotate it slightly, thereby increasing the angle.

So when the angle θ is maximized, we have $c = c'$ and thus $d(p, p') = d(q, q') = \delta\ell(s)/\sqrt{2}$. Then the segment $\overline{cp'}$ must be tangent to the circle centered at p with radius $\delta\ell(s)/\sqrt{2}$, hence $\theta = \arcsin(\sqrt{2}\delta)$. (See Figure 8.) □

2.4 Approximating the Hausdorff distance

In this section, we give algorithms for approximating the Hausdorff distance between two sets of segments S and S' in $\mathbb{R}^d$. We defined this distance $d_H(S, S')$ as the minimum value of δ such that, for all $s \in S$, there exists $s' \in S'$ satisfying $d(s, s') \leqslant \delta\ell(s)$.

2.4.1 Static case

We first give an algorithm for the static case, where S is not subjected to any transformation. Our algorithm begins with recording the segments of S', regarded as points in $\mathbb{R}^{2d}$, in the approximate near neighbor data structure (ANN) by Arya et al. [4]. It is constructed in $O(n \log n)$ time, and allows to report in time $O((1/\varepsilon^{2d}) \log n)$ a $(1+\varepsilon)$-approximate nearest segment in S'. In other words, it returns a segment $N^\varepsilon(s) \in S'$ that satisfies $d(s, N^\varepsilon(s)) \leqslant (1+\varepsilon)\min_{s' \in S'} d(s, s')$.

Then we compute $N^\varepsilon(s)$ for each segment $s \in S$, and we return $\delta^\varepsilon = \max_{s \in S} d(s, N^\varepsilon(s))/\ell(s)$. By the definition of approximate near neighbors, δ^ε is a $(1+\varepsilon)$-approximation of $d_H(S, S')$. It follows that:

Theorem 2. *If S and S' are sets of respectively m and n segments in $\mathbb{R}^d$, we can find a tolerance δ^ε such that $d_H(S, S') \leqslant \delta^\varepsilon \leqslant (1+\varepsilon) d_H(S, S')$ in time $O((m/\varepsilon^{2d} + n) \log n)$.*

If the the ANN data structure for S' was precomputed, the time bound becomes $O((m/\varepsilon^{2d}) \log n)$.

The second part of this theorem will be used in our algorithms for Hausdorff distance under translation and rotations: We will compute the ANN data structure for S' only once and reuse it each time we call the algorithm above on a translated or rotated copy of S.

2.4.2 Hausdorff distance under translation

Now we allow translations of S, and we want to minimize its Hausdorff distance to S'. We identify each translation with a point $\tau \in \mathbb{R}^d$. So we denote by $\tau + s$ the copy of s translated by vector τ, and $\tau + S$ denotes the set $\{\tau + s_1, \dots, \tau + s_m\}$. We denote by τ^* an optimal translation, in the sense that $d_H(\tau^* + S, S') = \min_{\tau \in \mathbb{R}^d} d_H(\tau + S, S')$. The Hausdorff distance under translation $d_H(\tau^* + S, S')$ is denoted by δ^* in this section, and we want to find a translation τ^ε that provides a $(1+\varepsilon)$-approximation of δ^*.

When we apply a translation τ to a segment, then the corresponding point in $\mathbb{R}^{2d}$ is translated by the vector (τ, τ) that has norm $\sqrt{2}\|\tau\|$. It implies the following.

Proposition 3. *For any two segments s, s' and translation τ, we have $d(\tau + s, s') \leqslant d(s, s') + \sqrt{2}\|\tau\|$.*

Without loss of generality, we assume that s_1 is a shortest segment in S, that is, $\ell_1 = \min_i \ell_i$. The lemma below bounds the variation of the Hausdorff distance after translating a set of segments.

Lemma 4. *For any translation τ, we have $d_H(\tau + S, S') \leqslant d_H(S, S') + \sqrt{2}\|\tau\|/\ell_1$.*

Proof. Proposition 3 implies that $d(\tau + s_i, s'_j) \leqslant d(s_i, s'_j) + \sqrt{2}\|\tau\|$ for all i, j. As s_1 is a shortest segment in S, it follows that $d(\tau + s_i, s'_j)/\ell_i \leqslant d(s_i, s'_j)/\ell_i + \sqrt{2}\|\tau\|/\ell_1$ for all i, j. In particular, for any $s_i \in S$, $\min_j d(\tau + s_i, s'_j)/\ell_i \leqslant \min_j d(s_i, s'_j)/\ell_i + \sqrt{2}\|\tau\|/\ell_1$. By definition of $d_H(S, S')$, there is a segment $s' \in S'$ at distance at most $d_H(S, S')\ell_i$ from any s_i, therefore

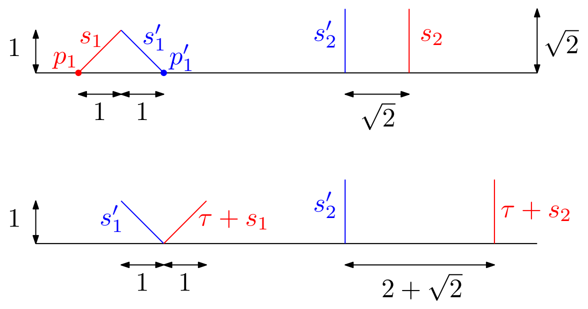


Figure 9: Worst-case example for Lemma 5.

$\min_j d(s_i, s'_j) \leqslant d_H(S, S')\ell_i$ and thus $\min_j d(\tau + s_i, s'_j)/\ell_i \leqslant d_H(S, S') + \sqrt{2}\|\tau\|/\ell_1$. It means that any segment $\tau + s_i$ has a segment of S' at distance at most $\ell_i(d_H(S, S') + \sqrt{2}\|\tau\|/\ell_1)$, in other words $d_H(\tau + S, S') \leqslant d_H(S, S') + \sqrt{2}\|\tau\|/\ell_1$. □

The bound in Lemma 4 is tight, in the sense that there exists sets of line segments S, S' and a translation τ such that $d_H(\tau + S, S') = d_H(S, S') + \sqrt{2}\|\tau\|/\ell_1$. It suffices to take $S = S' = \{s_1\}$, and to apply an arbitrary translation τ. Then $d_H(S, S') = 0$ and $d_H(\tau + S, S') = \sqrt{2}||\tau||/\ell_1$.

We now present a 3-approximation algorithm. For all j, let τ_j be the translation that maps p_1 to p'_j, in other words $\tau_j = p'_j - p_1$. The lemma below shows that one of these translations yields a $(1+\sqrt{2})$-approximation of δ^*.

Lemma 5. *Let* $\hat{\jmath} = \text{argmin}_{j=1,\dots,n} d_H(\tau_j + S, S')$. *Then* $d_H(\tau_{\hat{\jmath}} + S, S') \leqslant (1+\sqrt{2})\delta^*$.

Proof. The optimal translation τ^* matches s_1 with a segment $s'_k \in S'$ such that $d(\tau^* + s_1, s'_k) \leqslant \delta^*\ell_1$. It implies that $\|p'_k - p_1 - \tau^*\| \leqslant \delta^*\ell_1$, in other words $\|\tau_k - \tau^*\| \leqslant \delta^*\ell_1$. If follows from Lemma 4 that

$$\begin{aligned} d_H(\tau_k + S, S') &\leqslant d_H(\tau^* + S, S') + \sqrt{2}\|\tau_k - \tau^*\|/\ell_1 \\ &\leqslant d_H(\tau^* + S, S') + \sqrt{2}\delta^* \\ &= (1+\sqrt{2})\delta^*. \end{aligned}$$

□

The bound in Lemma 5 is also tight. To see this, consider the example in Figure 9. The length of each segment is $\sqrt{2}$. The optimal matching is shown on top: We have $d(s_1, s'_1) = d(s_2, s'_2) = 2$ and thus $\delta^* = 2/\sqrt{2} = \sqrt{2}$. After applying the horizontal translation τ_1 (bottom), we have $d(\tau_1 + s_1, s'_1) = 2$ and $d(\tau_1 + s_2, s'_2) = \sqrt{2}(2+\sqrt{2})$ so $d_H(\tau_1 + S, S') = 2 + \sqrt{2} = (1+\sqrt{2})\delta^*$.

We obtain a 6/5-approximation of $d_H(\tau_{\hat{\jmath}} + S, S')$ by running the algorithm of Theorem 2 for each τ_j, with $\varepsilon = 1/5$, and returning the best result. As $(1+\sqrt{2}) \cdot 6/5 < 3$, it gives us a 3-approximation.

Lemma 6. *Given* S *and* S', *we can compute in time* $O(mn \log n)$ *a translation* τ *and a tolerance* δ_3 *such that* $d_H(\tau + S, S') \leqslant \delta_3 \leqslant 3\delta^*$.

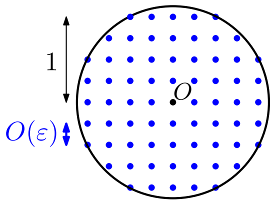


Figure 10: Sampling grid points on a unit ball.

We now provide a $(1+\varepsilon)$-approximation algorithm. We will discretize the space of translations based on the following discretization of the ball centered at the origin with radius $(1+\varepsilon)$. We use a uniform grid, so we let B^ε denote the set of points in this ball whose coordinates are multiples of $\varepsilon/\sqrt{d}$. More generally, for any $\lambda > 0$, we denote $\lambda B^\varepsilon = \{\lambda x \mid x \in B^\varepsilon\}$. (Figure 10.) This set contains $O(1/\varepsilon^d)$ points and can be constructed in constant time per point:

Proposition 7. *We can construct in time $O(1/\varepsilon^d)$ a set B^ε of $O(1/\varepsilon^d)$ points such that for any point p at distance at most 1 from the origin, there is a point $p' \in B^\varepsilon$ such that $\|p - p'\| \leqslant \varepsilon$.*

Our $(1+\varepsilon)$-approximation algorithm first computes the 3-approximation δ_3 of δ^* using Lemma 6, in other words we have $\delta^* \leqslant \delta_3 \leqslant 3\delta^*$. For each segment s'_j, it then constructs the set of translations $\mathcal{T}_j^\varepsilon = p'_j - p_1 + \delta_3 \ell_1 B^{\varepsilon/9}$, and then $\mathcal{T}^\varepsilon = \bigcup_j \mathcal{T}_j^\varepsilon$. (See Figure 7a.) We now prove that one translation $\hat{\tau} \in \mathcal{T}^\varepsilon$ gives a $(1+\varepsilon/2)$-approximation of the Hausdorff distance under translation.

Lemma 8. *There is a translation $\hat{\tau} \in \mathcal{T}^\varepsilon$ such that $d_H(\hat{\tau} + S, S') \leqslant (1+\varepsilon/2)\delta^*$.*

Proof. The optimal translation τ^* matches s_1 with a segment s'_k such that $d(\tau^* + s_1, s'_k) \leqslant \delta^* \ell_1$. So $\tau^* + p_1$ is at distance at most $\delta^* \ell_1$ from p'_k, in other words $\|\tau^* + p_1 - p'_k\| \leqslant \delta^* \ell_1 \leqslant \delta_3 \ell_1$. So there is a point $b^\varepsilon \in \delta_3 \ell_1 B^{\varepsilon/9}$ such that $\|\tau^* + p_1 - p'_k - b^\varepsilon\| \leqslant \delta_3 \ell_1 \varepsilon/9$. We let $\hat{\tau} = p'_k - p_1 + b^\varepsilon$, then $\hat{\tau} \in \mathcal{T}_k^\varepsilon \subset \mathcal{T}^\varepsilon$, and $\|\hat{\tau} - \tau^*\| \leqslant \delta_3 \ell_1 \varepsilon/9$. By Lemma 4, it implies that

$$\begin{aligned} d_H(\hat{\tau} + S, S') &\leqslant d_H(\tau^* + S, S') + \sqrt{2}\delta_3 \ell_1 \varepsilon/(9\ell_1) \\ &= \delta^* + \sqrt{2}\varepsilon\delta_3/9 \\ &\leqslant \delta^* + 3\sqrt{2}\varepsilon\delta^*/9 < (1+\varepsilon/2)\delta^*. \end{aligned}$$

□

In summary, we first compute the sample set of translations $\mathcal{T}^\varepsilon$. It consists of $O(n/\varepsilon^d)$ translations and can be constructed in $O(n/\varepsilon^d)$ time. For each translation $\tau \in \mathcal{T}^\varepsilon$, we compute in $O((m/\varepsilon^{2d})\log n)$ time a $(1+\varepsilon/3)$-approximation $\delta^\varepsilon(\tau)$ of $d_H(\tau + S, S')$ using Theorem 2. Let τ^ε be the translation that minimizes $\delta^\varepsilon(\tau^\varepsilon)$. Then we have $d_H(\tau^\varepsilon + S, S') \leqslant \delta^\varepsilon(\tau^\varepsilon) \leqslant \delta^\varepsilon(\hat{\tau}) \leqslant (1+\varepsilon/3) d_H(\hat{\tau} + S, S')$. By Lemma 8, it implies $d_H(\tau^\varepsilon + S, S') \leqslant (1+\varepsilon/2)(1+\varepsilon/3)\delta^* \leqslant (1+\varepsilon)\delta^*$. It completes the proof of the theorem below.

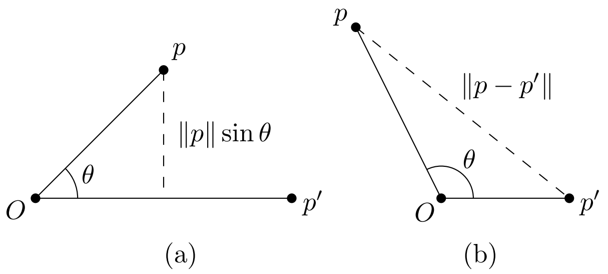


Figure 11: Proof of Lemma 10.

Theorem 9. *Let $\delta^* = d_H(\tau^* + S, S')$ be the Hausdorff distance between S and S' under translation. We can find a translation τ^ε and a tolerance δ^ε such that $d_H(\tau^\varepsilon + S, S') \leqslant \delta^\varepsilon \leqslant (1+\varepsilon)\delta^*$ in $O((mn/\varepsilon^{3d})\log n)$ time.*

2.4.3 Hausdorff distance under rotation about a fixed center in $\mathbb{R}^2$

In this section, we consider rotations about a fixed center in $\mathbb{R}^2$. Without loss of generality, we assume it to be the origin O. The rotation about O through an angle θ is denoted by ρ_θ. The optimal rotation ρ^* satisfies $d_H(\rho^*(S), S') = \min_{\theta\in[0,2\pi]} d_H(\rho_\theta(S), S')$, and we denote by θ^* its angle. The Hausdorff distance under rotation $d_H(\rho^*(S), S')$ is denoted by δ^* in this section, and we want to find a rotation ρ^ε that provides a $(1+\varepsilon)$-approximation of δ^*.

The distance from O to a point $p \in \mathbb{R}^2$ is denoted by $\|p\|$. We will need the following lemma.

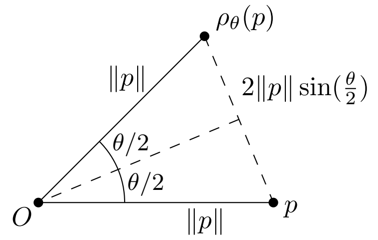
Lemma 10. *Let $p, p' \in \mathbb{R}^2$ be two points making an angle $\theta = \angle pOp'$ with the origin. If $\theta \in [-\pi, \pi]$, then $\|p - p'\| \geqslant \|p\| \cdot |\theta|/\pi$.*

Proof. If $|\theta| \leqslant \pi/2$, then by concavity of $\sin(\cdot)$ over $[0, \pi/2]$, we have $\sin(|\theta|) \geqslant 2|\theta|/\pi$. The distance between p and p' is at least the distance between p and its projection onto the line through O and p'. (See Figure 11a.) It follows that $\|p-p'\| \geqslant \|p\|\sin(|\theta|) \geqslant 2\|p\|\cdot|\theta|/\pi$. On the other hand, if $\pi/2 \leqslant |\theta| \leqslant \pi$, then $\|p-p'\| \geqslant \|p\|$, and thus $\|p-p'\| \geqslant \|p\|\cdot|\theta|/\pi$. (See Figure 11b.) □

The *aspect ratio* of a segment $s = (p, q)$ is $\alpha(s) = \max(\|p\|, \|q\|)/\ell(s)$. When we apply the rotation ρ_θ to s, then p and q move by a distance $2\sin(|\theta/2|)\|p\|$ and $2\sin(|\theta/2|)\|q\|$, respectively. (See Figure 12.) Therefore, each endpoint p or q moves by a distance at most $2\sin(|\theta/2|)\max(\|p\|,\|q\|) = 2\sin(|\theta/2|)\ell(s)\alpha(s) \leqslant |\theta|\ell(s)\alpha(s)$. So regarded as a point in $\mathbb{R}^{2d}$, the segment s is translated by a vector of norm at most $\sqrt{2}|\theta|\ell(s)\alpha(s)$, and thus:

Proposition 11. *For any two segments s and s', and for any angle θ, we have $d(\rho_\theta(s), s') \leqslant d(s, s') + \sqrt{2}|\theta|\ell(s)\alpha(s)$.*

Let $s_a = (p_a, q_a)$ be the segment in S with largest aspect ratio. So we have $\alpha(s_a) = \max_{s\in S}\alpha(s)$ and $\ell_a = \ell(s_a)$, and we denote by $\alpha_a = \alpha(s_a)$ its aspect ratio. Without loss of generality, we assume that $\|p_a\| \geqslant \|q_a\|$, and thus $\alpha_a = \|p_a\|/\ell_a$.



(c)

Figure 12: Proof of Proposition 11.

Lemma 12. *$d_H(\rho_\theta(S), S') \leqslant d_H(\rho_{\theta'}(S), S') + \sqrt{2}\alpha_a|\theta - \theta'|$ for any two angles θ and θ'.*

Proof. By Proposition 11, we have for any two segments $s \in S$ and $s' \in S'$

$$\frac{d(\rho_{\theta-\theta'}(\rho_{\theta'}(s)), s')}{\ell(s)} \leqslant \frac{d(\rho_{\theta'}(s), s')}{\ell(s)} + \sqrt{2}|\theta - \theta'|\alpha(\rho_{\theta'}(s)).$$

As $\rho_{\theta-\theta'}(\rho_{\theta'}(s)) = \rho_\theta(s)$, $\alpha(\rho_{\theta'}(s)) = \alpha(s)$ and $\alpha(s) \leqslant \alpha_a$, it implies

$$\begin{aligned}\frac{d(\rho_\theta(s), s')}{\ell(s)} &\leqslant \frac{d(\rho_{\theta'}(s), s')}{\ell(s)} + \sqrt{2}|\theta - \theta'|\alpha(s)\\ &\leqslant \frac{d(\rho_{\theta'}(s), s')}{\ell(s)} + \sqrt{2}|\theta - \theta'|\alpha_a.\end{aligned}$$

The result follows directly from our definition of the Hausdorff distance between sets of segments.

□

We now present a 6-approximation algorithm for δ^*. For all j, let θ_j denote the rotation angle such that p_a lies on the ray from O to p'_j. (See Figure 7b.) If $p'_j = O$, then we can choose $\theta = 0$. The lemma below shows that one of these angles gives a $(\sqrt{2}\pi + 1)$-approximation.

Lemma 13. *Let $\hat{j} = \operatorname{argmin}_{j=1,\dots,n} d_H(\rho_{\theta_j}(S), S')$. Then $d_H(\rho_{\theta_{\hat{j}}}(S), S') \leqslant (\sqrt{2}\pi + 1)\delta^*$.*

Proof. The optimal rotation ρ^* matches s_a with a segment $s'_j \in S'$ such that $d(\rho^*(s_a), s'_j) \leqslant \delta^*\ell_a$. It implies that $\|\rho^*(p_a) - p'_j\| \leqslant \delta^*\ell_a$. Let $\theta = \theta_j - \theta^*$. In other words, we have $\theta = \angle\rho^*(p_a)Op'_j$. By Lemma 10, we have $\|\rho^*(p_a) - p'_j)\| \geqslant \|\rho^*(p_a)\| \cdot |\theta|/\pi$, and thus $\delta^*\ell_a \geqslant \|\rho^*(p_a)\| \cdot |\theta|/\pi$. As $\|\rho^*(p_a)\| = \|p_a\|$, it means that $\pi\delta^* \geqslant |\theta|\alpha_a$. Then Lemma 12 yields

$$\begin{aligned}d_H(\rho_{\theta_j}(S), S') &\leqslant d_H(\rho_{\theta^*}(S), S') + \sqrt{2}|\theta|\alpha_a\\ &\leqslant d_H(\rho_{\theta^*}(S), S') + \sqrt{2}\pi\delta^*\\ &= (\sqrt{2}\pi + 1)\delta^*.\end{aligned}$$

□

We obtain our 6-approximation of δ^* by applying the algorithm of Theorem 2 with $\varepsilon = (6/(\sqrt{2}\pi + 1)) - 1$ to the pair of sets $\rho_{\theta_j}(S)$, S' for $j = 1, \dots, n$, and returning the best result.

Lemma 14. *Given S, S', we can compute in time $O(mn \log n)$ an angle θ and a tolerance δ_6 such that $d_H(\rho_\theta(S), S') \leqslant \delta_6 \leqslant 6\delta^*$.*

We now present our $(1+\varepsilon)$-approximation algorithm. We begin with computing the 6-approximation δ_6 from Lemma 14, and then we discretize the set of rotation angles. To this end, we observe that the optimal angle θ^* must be close to an angle θ_j.

Lemma 15. *There exists $j \in \{1, \dots, n\}$ such that $|\theta^* - \theta_j| \leqslant \pi\delta_6/\alpha_a$.*

Proof. The optimal rotation ρ^* must bring $\rho^*(p_a)$ to a distance at most $\delta^*\ell_a$ of a point p'_j, and thus $\|\rho^*(p_a) - p'_j\| \leqslant \ell_a\delta^*$. As $\angle\rho^*(p_a)Op'_j = |\theta_j - \theta^*|$, Lemma 10 implies that $\|\rho^*(p_a)\| \cdot |\theta^* - \theta_j|/\pi \leqslant \|\rho^*(p_a) - p'_j\|$. As $\|\rho^*(p_a)\| = \|p_a\|$, it yields $\|p_a\| \cdot |\theta^* - \theta_j|/\pi \leqslant \delta^*\ell_a$. The result follows from the facts that $\alpha_a = \|p_a\|/\ell_a$ and $\delta^* \leqslant \delta_6$. □

For each $j \in \{1, \dots, n\}$, we denote by Θ^ε_j a set of $O(1/\varepsilon)$ angles sampled uniformly in the interval $[\theta_j - \pi\delta_6/\alpha_a, \theta_j + \pi\delta_6/\alpha_a]$. More precisely,

$$\Theta^\varepsilon_j = \{\theta_j + kC_1\varepsilon\delta_6/\alpha_a \mid k \in \mathbb{Z} \text{ and } |kC_1\varepsilon| < \pi\},$$

where C_1 is a constant to be determined. Let $\Theta^\varepsilon = \bigcup_j \Theta^\varepsilon_j$ be the union of these n sets. By Lemma 15, there is an angle $\hat\theta \in \Theta^\varepsilon$ such that $|\hat\theta - \theta^*| \leqslant C_1\varepsilon\delta_6/\alpha_a$. It follows from Lemma 12 that

$$\begin{aligned} d_H(\rho_{\hat\theta}(S), S') &\leqslant d_H(\rho_{\theta^*}(S), S') + \sqrt{2}\alpha_a|\hat\theta - \theta^*| \\ &\leqslant d_H(\rho_{\theta^*}(S), S') + \sqrt{2}C_1\varepsilon\delta_6 \\ &\leqslant d_H(\rho_{\theta^*}(S), S') + 6\sqrt{2}C_1\varepsilon\delta^* \\ &= (1 + 6\sqrt{2}C_1\varepsilon)\delta^*. \end{aligned}$$

Choosing $C_1 = \sqrt{2}/24$, we obtain the following discretization.

Lemma 16. *Given δ_6, we can compute in time $O(n/\varepsilon)$ a set Θ^ε of $O(n/\varepsilon)$ angles such that one of these angles $\hat\theta$ satisfies $d_H(\rho_{\hat\theta}(S), S') \leqslant (1 + \varepsilon/2)\delta^*$.*

For each angle in this set, we run the static algorithm of Theorem 2 using an approximation factor $(1+\varepsilon/3)$, and keep the best tolerance δ^ε. As $\varepsilon < 1$, we have $(1+\varepsilon/3)(1+\varepsilon/2) < 1+\varepsilon$, hence we obtain a $(1+\varepsilon)$-approximation.

Theorem 17. *Given S and S', we can compute in time $O((mn/\varepsilon^5)\log n)$ an angle $\hat\theta$ and a tolerance δ^ε such that $d_H(\rho_{\hat\theta}(S), S') \leqslant \delta^\varepsilon \leqslant (1+\varepsilon)\delta^*$.*

2.4.4 Hausdorff distance under rigid motion in $\mathbb{R}^2$

In this section, our goal is to approximate the Hausdorff distance under rigid motions between S and S', so we allow to rotate and translate S. We will not consider glided reflections: they can be handled by considering separately S and one reflected copy of S as inputs to our algorithm for compositions of rotations and translations. Our approach is to combine the approximation algorithms of the two previous sections for the translation case and for rotations about a fixed center.

We denote by $\mathcal{R}$ our set of rigid motions in $\mathbb{R}^2$, which means translations and rotations about an arbitrary center. Equivalently, a rigid motion is the composition of a rotation about a fixed center with a translation. The optimal rigid motion μ^* satisfies

$$d_H(\mu^*(S), S') = \min_{\mu \in \mathcal{R}} d_H(\mu(S), S').$$

The Hausdorff distance under rigid motion $d_H(\mu^*(S), S')$ is denoted by δ^* in this section, and we want to find a rigid motion μ^ε that provides a $(1+\varepsilon)$-approximation of δ^*.

We denote by $\rho_{p,\theta}$ the rotation about a point p through an angle θ. We denote by $\mu_{\tau,\theta}$ the rigid motion consisting of the rotation about p_1 through an angle θ, followed by the translation τ. Hence we have

$$\mu_{\tau,\theta} = \tau \circ \rho_{p_1,\theta} = \rho_{\tau+p_1,\theta} \circ \tau. \tag{1}$$

Let $S_\theta = \rho_{p_1,\theta}(S)$ be the set S rotated through an angle θ about p_1. Then it follows from Equation (1) that

$$\delta^* = \min_{\theta \in [0,2\pi]} \left(\min_{\tau \in \mathbb{R}^2} d_H(\tau + S_\theta, S') \right) \tag{2}$$

and if we denote by τ^* and θ^* the parameters of the optimal translation μ^* (hence $\mu^* = \mu_{\tau^*,\theta^*}$) we have

$$\delta^* = d_H(\tau^* + S_{\theta^*}, S') = \min_{\tau \in \mathbb{R}^2} d_H(\tau + S_{\theta^*}, S'). \tag{3}$$

For any fixed angle θ, Lemma 5 shows that a translation $\tau_j = p'_j - p_1$ gives a $(1+\sqrt{2})$-approximation of the optimal translation. So Equation (2) implies that there exists $j \in \{1, \dots, n\}$ such that $\min_{\theta \in [0,2\pi]} d_H(\tau_j + S_\theta, S') \leqslant (1+\sqrt{2})\delta^*$. By Equation (1), it means that there exists $j \in \{1, \dots, n\}$ such that

$$\min_{\theta \in [0,2\pi]} d_H(\mu_{\tau_j,\theta}(S), S') \leqslant (1+\sqrt{2})\delta^*. \tag{4}$$

In other words, there is a $(1+\sqrt{2})$-approximate solution $\mu_{\tau,\theta}$ whose translation part is $\tau = \tau_j$ for some j. Therefore, we can find a constant approximation by trying all translations τ_j, and then minimizing over θ for each of them. We perform this minimization using our approximation algorithm for rotation about a fixed center (Theorem 17), with $\varepsilon = 1/5$ and the center being $p'_j = \tau_j + p_1$, thus obtaining a 3-approximation.

Lemma 18. *We can compute in time $O(mn^2 \log n)$ a tolerance δ'_3 such that $\delta^* \leqslant \delta'_3 \leqslant 3\delta^*$.*

Similarly as what we did for translations and rotations about a fixed center, we make use of this constant-factor approximation to obtain a $(1+\varepsilon)$-approximation. So we first compute the same set of candidate translations $\mathcal{T}^\varepsilon$ as in Section 2.4.2, except that we replace δ_3 with δ'_3: we now have $\mathcal{T}_j^\varepsilon = p'_j - p_1 + \delta'_3 \ell_1 B^{\varepsilon/9}$ and $\mathcal{T}^\varepsilon = \bigcup_j \mathcal{T}_j^\varepsilon$, where $B^{\varepsilon/9}$ is our discretization of the unit ball using grid points.

As δ'_3 is a 3-approximation of the Hausdorff distance under translation between S_{θ^*} and S', Lemma 8 shows that there is a translation $\hat{\tau} \in \mathcal{T}^\varepsilon$ such that $d(\hat{\tau} + S_{\theta^*}, S') \leqslant (1+\varepsilon/2)\delta^*$. So our approximation algorithm for rotations about a fixed center (Theorem 17) applied to the sets $\hat{\tau} + S, S'$, using approximation factor $(1+\varepsilon/3)$ and center $\hat{\tau} + p_1$, yields a $(1+\varepsilon)$-approximation of δ^*. As we don't know $\hat{\tau}$ in advance, we apply this algorithm to all the translations in $\mathcal{T}^\varepsilon$ and keep the best solution. As we are applying Theorem 17 to the $O(n/\varepsilon^2)$ translations in $\mathcal{T}^\varepsilon$, it takes time $O((mn^2/\varepsilon^7)\log n)$.

Theorem 19. *Given S and S', we can compute in time $O((mn^2/\varepsilon^7)\log n)$ a rigid motion μ^ε and a tolerance δ^ε such that $d_H(\mu^\varepsilon(S), S') \leqslant \delta^\varepsilon \leqslant (1+\varepsilon)\delta^*$.*

2.5 Approximating the bottleneck distance

In this section, we present modified versions of the algorithms from the previous sections that match segments of S and S' according to the bottleneck distance, instead of the Hausdorff distance. This distance differs from the Hausdorff distance in that it requires the matching between segments of S and S' to be one-to-one. More precisely, we define the bottleneck distance $d_b(S, S')$ between two sets of segments as follows: $d_b(S, S')$ is the smallest tolerance δ such that there exists a one-to-one mapping $\sigma : S \to S'$ with $d(s, \sigma(s)) \leqslant \delta\ell(s)$ for all $s \in S$.

So instead of handling each segment of S by making an ANN query, as we did in the previous sections on Hausdorff distance, we need to find a matching of cardinality $|S| = m$ in the bipartite graph over $S \cup S'$, where two segments s, s' are connected by an edge if $d(s, s') \leqslant \delta\ell(s)$, for some tolerance δ. Efrat et al. [5] showed how to compute such matchings efficiently for point sets in $\mathbb{R}^d$. We obtain below similar results for sets of line segments by combining their techniques with the discretization schemes that we presented in Section 2.4.

2.5.1 Static case

We first consider the static case, so we want to approximate $d_b(S, S')$ without any transformation of S. We now show how to adapt the result by Efrat et al. to our new setting. Let us give its statement [5, Theorem 7.3], specialized to the case $d = O(1)$, $0 < \varepsilon < 1$ and for the L_2 norm:

Theorem 20 (Efrat et al.). *Let A and B be sets of n points in $\mathbb{R}^d$. Let r^* be the bottleneck distance between A and B. We can find in time $O((n^{1.5}/\varepsilon^d)\log n)$ a matching between A and B whose longest edge has length at most $(1+\varepsilon)r^*$.*

Their approach is the following. They first design an *approximating oracle* which, given a query radius r, returns a positive answer if $(1+\varepsilon)r \geqslant r^*$, and otherwise $r < r^*$. This oracle searches for a matching in a bipartite graph over $A \times B$, where an edge is drawn between two points if their distance is at most r, and no edge is drawn if their distance is more than $(1+\varepsilon)r$.

The edges of this graph are not all constructed, but they are constructed on the fly by making ANN queries using Arya et al. [4] data structure: If the ANN is at distance at most $r(1+\varepsilon)$, then a new edge is discovered. This explains why the running time dependency on n is only $O(n^{1.5})$, and the dependency on ε is $O(1/\varepsilon^d)$. In our setting, the segments are regarded as points in $\mathbb{R}^{2d}$ so the dependency on ε becomes $1/\varepsilon^{2d}$, and the threshold for discovering a new edge (s, s') becomes $\ell(s)\delta(1+\varepsilon)$. So the algorithm is the same, and we just increase the dependency of ε from $1/\varepsilon^d$ to $1/\varepsilon^{2d}$.

Efrat et al. then use this approximating oracle to approximate the optimal radius r^* by binary search. In order to reduce the search interval, they first compute a constant-factor approximation by finding an approximate optimal matching under the L_∞ metric. The point here is that an optimal, or a $(1+\varepsilon)$-approximate, bottleneck matching can be found using the sorted matrix searching technique, as the optimal radius is the difference between two coordinates of two input points. Using the technique by Frederickson and Johnson [17] for selection in sorted matrices (that is, matrices whose rows and columns are sorted), it suffices to perform $O(\log n)$ steps in the binary search, and the running time is dominated by the calls to the oracle.

In our case, as the distance $d(s, s')$ is weighted by $\ell(s)$, we do not have a sorted matrix in the sense that rows and columns are sorted. However, for each segment s_i, the candidate tolerances δ are of the form $d(s_i, s'_j)/\ell(s_i)$. So the whole matrix of candidate tolerances consists of $2d$ sorted rows. Therefore, after transposing the matrix, we can employ another technique by Frederickson and Johnson [18, Theorem 1] for matrices with sorted columns, and the running time is still dominated by $O(\log n)$ calls to the approximating oracle.

In summary, we obtain the following result:

Theorem 21. *We can compute in time $O((n^{1.5}/\varepsilon^{2d})\log n)$ a tolerance δ^ε such that $d_b(S, S') \leqslant \delta^\varepsilon \leqslant (1+\varepsilon)d_b(S, S')$, and a matching M^ε between S and S' that achieves this tolerance in the sense that for any edge $(s, s') \in M^\varepsilon$, we have $d(s, s') \leqslant \delta^\varepsilon \ell(s)$.*

2.5.2 Bottleneck distance under translation

The result above can be used to approximate the bottleneck distance under translation. We apply the same approach as in Section 2.4, except that we employ our algorithm for bottleneck distance (Theorem 21) instead of the algorithm for Hausdorff distance (Theorem 2). We now give a detailed proof for this case, that is, for bottleneck distance under translation. It closely follows the proof for Hausdorff distance under translation.

Our goal is to approximate the optimal translation τ^* such that $d_b(\tau^* + S, S') = \min_{\tau\in\mathbb{R}^d} d_b(\tau + S, S')$. So in this section, we denote by $\delta^* = d_b(\tau^* + S, S')$ the bottleneck distance under translation, and we denote by σ^* a one-to-one mapping that achieves it, which means that $d(\tau^* + s, \sigma^*(s)) \leqslant \ell(s)\delta^*$ for all $s \in S$. We first prove a lemma analogous to 4. Remember that s_1 is the shortest segment in S.

Lemma 22. *For any translation τ, we have $d_b(\tau + S, S') \leqslant d_b(S, S') + \sqrt{2}\|\tau\|/\ell_1$.*

Proof. By definition of the bottleneck distance, there is a one-to-one mapping σ such that $d(s,\sigma(s)) \leqslant d_b(S,S')\ell(s)$ for each $s \in S$. Proposition 3 implies that $d(\tau+s,\sigma(s)) \leqslant d(s,\sigma(s)) + \sqrt{2}\|\tau\|$ for all $s \in S$. As s_1 is a shortest segment in S, it follows that

$$\frac{d(\tau+s,\sigma(s))}{\ell(s)} \leqslant \frac{d(s,\sigma(s))}{\ell(s)} + \sqrt{2}\frac{\|\tau\|}{\ell_1}$$

for all $s \in S$. As $d(s,\sigma(s)) \leqslant d_b(S,S')\ell(s)$, it implies that $d_b(\tau+S,S') \leqslant d_b(S,S')+\sqrt{2}\|\tau\|/\ell_1$. □

We now present a 3-approximation algorithm similar with our algorithm for Hausdorff distance. So for all j, we let $\tau_j = p'_j - p_1$, and we show that one of these translations yields a $(1+\sqrt{2})$-approximation of δ^*.

Lemma 23. *Let* $\hat{\jmath} = \operatorname{argmin}_{j=1,\dots,n} d_b(\tau_j + S, S')$. *Then* $d_b(\tau_{\hat{\jmath}} + S, S') \leqslant (1+\sqrt{2})\delta^*$.

Proof. The optimal translation τ^* matches s_1 with a segment $s'_k = \sigma^*(s_1)$ such that $d(\tau^* + s_1, s'_k) \leqslant \delta^*\ell_1$. It implies that $\|p'_k - p_1 - \tau^*\| \leqslant \delta^*\ell_1$, in other words $\|\tau_k - \tau^*\| \leqslant \delta^*\ell_1$. If follows from Lemma 22 that

$$\begin{aligned} d_b(\tau_k + S, S') &\leqslant d_b(\tau^* + S, S') + \sqrt{2}\|\tau_k - \tau^*\|/\ell_1 \\ &\leqslant d_b(\tau^* + S, S') + \sqrt{2}\delta^* \\ &= (1+\sqrt{2})\delta^*. \end{aligned}$$

□

We obtain a 6/5-approximation of $d_b(\tau_{\hat{\jmath}} + S, S')$ by running the algorithm of Theorem 21 for each τ_j, with $\varepsilon = 1/5$, and returning the best result. As $(1+\sqrt{2}) \cdot 6/5 < 3$, it gives us a 3-approximation of δ^*.

Lemma 24. *Given* S *and* S', *we can compute in time* $O(n^{2.5}\log n)$ *a translation* τ *and a tolerance* δ''_3 *such that* $d_b(\tau + S, S') \leqslant \delta''_3 \leqslant 3\delta^*$.

Our $(1+\varepsilon)$-approximation algorithm first computes the 3-approximation δ''_3 of δ^* using Lemma 24, in other words we have $\delta^* \leqslant \delta''_3 \leqslant 3\delta^*$. For each segment s'_j, it then constructs the set of translations $\mathcal{T}^\varepsilon_j = p'_j - p_1 + \delta''_3\ell_1 B^{\varepsilon/9}$, and then $\mathcal{T}^\varepsilon = \bigcup_j \mathcal{T}^\varepsilon_j$. (See Figure 7a.) We now prove that one translation $\hat{\tau} \in \mathcal{T}^\varepsilon$ gives a $(1+\varepsilon/2)$-approximation of the bottleneck distance under translation.

Lemma 25. *There is a translation* $\hat{\tau} \in \mathcal{T}^\varepsilon$ *such that* $d_b(\hat{\tau} + S, S') \leqslant (1+\varepsilon/2)\delta^*$.

Proof. The optimal translation τ^* maps s_1 to a segment $s'_k = \sigma^*(s_1)$ such that $d(\tau^*+s_1, s'_k) \leqslant \delta^*\ell_1$. So $\tau^* + p_1$ is at distance at most $\delta^*\ell_1$ from p'_k, in other words $\|\tau^* + p_1 - p'_k\| \leqslant \delta^*\ell_1 \leqslant \delta''_3\ell_1$. So there is a point $b^\varepsilon \in \delta''_3\ell_1 B^{\varepsilon/9}$ such that $\|\tau^* + p_1 - p'_k - b^\varepsilon\| \leqslant \delta''_3\ell_1\varepsilon/9$. We let $\hat{\tau} = p'_k - p_1 + b^\varepsilon$, then $\hat{\tau} \in \mathcal{T}^\varepsilon_k \subset \mathcal{T}^\varepsilon$, and $\|\hat{\tau} - \tau^*\| \leqslant \delta''_3\ell_1\varepsilon/9$. By Lemma 22, it implies that

$$\begin{aligned} d_b(\hat{\tau} + S, S') &\leqslant d_b(\tau^* + S, S') + \sqrt{2}\delta''_3\ell_1\varepsilon/(9\ell_1) \\ &= \delta^* + \sqrt{2}\varepsilon\delta''_3/9 \\ &\leqslant \delta^* + 3\sqrt{2}\varepsilon\delta^*/9 < (1+\varepsilon/2)\delta^*. \end{aligned}$$

□

In summary, as we did for the Hausdorff distance under translation, we first compute the sample set of translations $\mathcal{T}^\varepsilon$. It consists of $O(n/\varepsilon^d)$ translations and can be constructed in $O(n/\varepsilon^d)$ time. For each translation $\tau \in \mathcal{T}^\varepsilon$, we compute in $O((n^{1.5}/\varepsilon^{2d})\log n)$ time a $(1+\varepsilon/3)$-approximation $\delta^\varepsilon(\tau)$ of $d_b(\tau + S, S')$ using Theorem 21. Let τ^ε be the translation that minimizes $\delta^\varepsilon(\tau^\varepsilon)$. Then we have $d_b(\tau^\varepsilon + S, S') \leqslant \delta^\varepsilon(\tau^\varepsilon) \leqslant \delta^\varepsilon(\hat{\tau}) \leqslant (1+\varepsilon/3) d_b(\hat{\tau} + S, S')$. By Lemma 25, it implies $d_b(\tau^\varepsilon + S, S') \leqslant (1+\varepsilon/2)(1+\varepsilon/3)\delta^* \leqslant (1+\varepsilon)\delta^*$. It completes the proof of the theorem below.

Theorem 26. *Let $\delta^* = d_b(\tau^* + S, S')$ be the bottleneck distance between S and S' under translation. We can find a translation τ^ε and a tolerance δ^ε such that $d_b(\tau^\varepsilon + S, S') \leqslant \delta^\varepsilon \leqslant (1+\varepsilon)\delta^*$ in $O((n^{2.5}/\varepsilon^{3d})\log n)$ time.*

2.5.3 Bottleneck distance under rotation and rigid motion in $\mathbb{R}^2$

We can similarly combine Theorem 21 with our discretization of the set of rotations about a fixed center, and the set of rigid motions, obtaining approximation schemes for the bottleneck distance under these sets of transformations. For a fixed center, we consider $O(n/\varepsilon)$ rotations, and in the second case we consider $O(n^2/\varepsilon^3)$ rigid motions. We do not give detailed proofs as we did in the previous section; the same modifications of the algorithms for Hausdorff distance that we applied for translations can be carried out as well for rotations and rigid motions.

Theorem 27. *Given two sets of segments S and S' in $\mathbb{R}^2$, we can compute in time $O((n^{2.5}/\varepsilon^5)\log n)$ a $(1+\varepsilon)$-approximation of the bottleneck distance under rotation about a fixed center. For arbitrary rigid motions, the time bound increases to $O((n^{3.5}/\varepsilon^7)\log n)$*

2.6 Approximating the largest common subset

Given two sets of segments S and S' and a tolerance δ, the LCS problem is to find subsets $C \subset S$ and $C' \subset S'$ of largest cardinality $|C| = |C'|$ whose bottleneck distance $d_b(C, C')$ is at most δ. We denote by $\mathsf{LCS}(S, S', \delta)$ this optimal cardinality. In this section, we consider an approximate version where we want to find subsets of size at least $\mathsf{LCS}(S, S', \delta)$ whose bottleneck distance is at most $\delta(1+\varepsilon)$ for some $0 < \varepsilon < 1$. Our approach is very similar with our algorithms for the bottleneck distance.

We can reformulate this problem more concisely as follows. Let $G[\delta]$ denote the bipartite graph over $S \cup S'$ where s and s' are connected whenever $d(s, s') \leqslant \delta\ell(s)$. The LCS problem is to find a maximum matching in $G[\delta]$. The approximate version is to find a matching in $G[\delta(1+\varepsilon)]$ of size at least $\mathsf{LCS}(S, S', \delta)$.

2.6.1 Static case

The approximation algorithm by Efrat et al. [5, Theorem 7.3] finds a matching in $G[\delta(1+\varepsilon)]$ of size at least the size of a maximum matching in $G[\delta]$, so this is exactly what we need. Our time bound increases by a factor $(1/\varepsilon^d)$ because our segments are regarded as points in $\mathbb{R}^{2d}$, so the dependency on ε of the ANN data structure is $O(1/\varepsilon^{2d})$.

Theorem 28. *We can compute a $(1+\varepsilon)$-approximation of the LCS between two sets of segments in time $O((n^{1.5}/\varepsilon^{2d})\log n)$.*

2.6.2 LCS under translation

For LCS under translation, we apply this algorithm to a discrete set of translated versions $\tau + S$ of S. However, as opposed to what we did for Hausdorff distance and bottleneck distance, the shortest segment s_1 may not be part of an optimal matching: If $C \subset S$ and $C' \subset S'$ are optimal solutions to our problem, then we do not necessarily have $s_1 \in C$.

So we take a larger set of candidate translations, where each segment s_i plays the role that s_1 played in the Hausdorff case: $\mathcal{R}^{\varepsilon}_{i,j} = p'_j - p_i + \delta\ell_i B^{\varepsilon/3}$ and $\mathcal{R}^{\varepsilon} = \bigcup_{i,j} R^{\varepsilon}_{i,j}$. Thus, the number of candidate translations has increased by a factor m.

Let (C_*, C'_*) denote an optimal solution to our problem, so for an optimal translation τ^* we have $d_b(\tau^*+C_*, C'_*) \leqslant \delta$ and $|C_*| = |C'_*| = \mathsf{LCS}(\tau^*+C_*, C'_*, \delta)$. We denote by σ^* a corresponding matching, that is, $\sigma^* : C_* \to C'_*$ is a one-to-one mapping such that $d(s, \sigma^*(s)) \leqslant \delta\ell(s)$ for all $s \in C_*$.

Lemma 29. *There is a translation $\hat{\tau} \in \mathcal{R}^{\varepsilon}$ such that $\mathsf{LCS}(\hat{\tau}+S, S', \delta(1+\varepsilon/2)) \geqslant \mathsf{LCS}(\tau^*+S, S', \delta)$.*

Proof. Let s_i be a shortest segment in C_*, and let $s'_j = \sigma^*(s_i)$ be the segment that it is matched with in an optimal solution. In particular, we have $d(\tau^*+s_i, s'_j) \leqslant \delta\ell_i$ and thus $d(\tau^*+p_i, p'_j) \leqslant \delta\ell_i$. Therefore, there is a point b^{ε} in $\delta\ell_i B^{\varepsilon/3}$ such that $\|\tau^* + p_i - p'_j - b^{\varepsilon}\| \leqslant \delta\ell_i\varepsilon/3$.

We let $\hat{\tau} = p'_j - p_i + b^{\varepsilon}$, then $\hat{\tau} \in \mathcal{R}^{\varepsilon}_{i,j} \subset \mathcal{R}^{\varepsilon}$ and $\|\hat{\tau} - \tau^*\| \leqslant \delta\ell_i\varepsilon/3$. By Lemma 22, as ℓ_i is a shortest segment in C_*, it implies:

$$\begin{aligned} d_b(\hat{\tau} + C_*, C'_*) &\leqslant d_b(\tau^* + C_*, C'_*) + \frac{\sqrt{2}}{\ell_i}\|\hat{\tau} - \tau^*\| \\ &\leqslant \delta + \frac{\sqrt{2}}{\ell_i}\frac{\delta\ell_i\varepsilon}{3} < \delta\left(1 + \frac{\varepsilon}{2}\right). \end{aligned}$$

In particular, it means that

$$\begin{aligned} \mathsf{LCS}(\hat{\tau} + S, S', \delta(1+\varepsilon/2)) &\geqslant |C_*| \\ &= \mathsf{LCS}(\tau^* + S, S', \delta). \end{aligned}$$

□

Thus our algorithm for LCS under translation is the following: we first compute the set $\mathcal{R}^{\varepsilon}$ of $O((1/\varepsilon^d)mn)$ candidate translations, and for each translation $\tau \in \mathcal{R}^{\varepsilon}$, we run the static LCS algorithm with approximation factor $(1+\varepsilon/3)$ on $\tau + S$ and S'. We then return the best pair of subsets of S and S' that we obtained. It provide a $(1+\varepsilon/2)(1+\varepsilon/3) \leqslant (1+\varepsilon)$-approximation.

Theorem 30. *For segments under translation in $\mathbb{R}^d$, we can compute a $(1+\varepsilon)$-approximation of the LCS in time $O((mn^{2.5}/\varepsilon^{3d})\log n)$. More precisely, we find τ_ε, $C_\varepsilon \subset S$, and $C'_\varepsilon \subset S'$ such that $|C_\varepsilon| = |C'_\varepsilon| \geqslant \max_\tau \mathsf{LCS}(\tau + S, S', \delta)$ and $d_b(\tau_\varepsilon + C_\varepsilon, C'_\varepsilon) \leqslant \delta(1+\varepsilon)$.*

2.6.3 Rotations and rigid motions

The same approach as we used for translations can be applied to the cases of rotations about a fixed center and rigid motions in $\mathbb{R}^2$. We briefly sketch our algorithms and the results obtained.

In the case of rotations about a fixed center in $\mathbb{R}^2$, as the segments $s_a \in S$ with largest aspect ratio does not necessarily appear in the optimum matching, we let all the segments in S play the role that s_a played in the Hausdorff case. So the size of our discretization of the angles is $O(mn/\varepsilon)$, instead of $O(n/\varepsilon)$ for the Hausdorff case. (Lemma 16.) We then run the static algorithm for LCS (Theorem 28) after rotating S by each of these angles.

For arbitrary rigid motions, the size of the discretization is m^2 times larger than our discretization for the Hausdorff case, because we need to try m times as many translations and for each translation, m times as many angles. So we call the static LCS algorithms $O(m^2n/\varepsilon^3)$ times, which yields the following.

Theorem 31. *We can compute in time $O((mn^{2.5}/\varepsilon^5)\log n)$ a $(1+\varepsilon)$-approximation of the LCS under rotation about a fixed center in $\mathbb{R}^2$. For arbitrary rigid motions, the time bound becomes $O((m^2n^{3.5}/\varepsilon^7)\log n)$.*

2.7 Undirected line segments

As mentioned in the introduction, our results also apply to *undirected* line segments. For the Hausdorff case, it suffices to add all the reverse segments $r'_j = (q'_j, p'_j)$ to S', and then the algorithms are unchanged. For the bottleneck and LCS cases, we also add the reverse segments, and we need a simple modification of Efrat et al. [5] algorithm for geometric matchings: after inserting an edge (s_i, s'_j) or (s_i, r'_j) in the matching, we need to delete from the ANN data structure the reverse segment r'_j or s'_j, respectively. It ensures that r'_j and s'_j do not both appear in the matching.

2.8 Lower bound

We show that matching sets of line segments under translation with respect to the Hausdorff distance, is in the class of 3SUM-hard (see definition 2) problems, introduced by Gajentaan and Overmars [19]. To explain our strategy, we introduce 3SUM, which is a well-known problem for which no subquadratic algorithm is known, and another problem called segments containing points (SegContPnt) which is very similar to our problem and 3sum-hard.

Definition 1. *(definition in [20]) Given two problems PR1 and PR2 we say that PR1 is f(n)-solvable using PR2 if every instance of PR1 of size n can be solved by using a constant number of instances of PR2 (of size O(n)) and O(f(n)) additional time. We denote this as*

$$PR1 \lll_{f(n)} PR2 \qquad .$$

Problem 1. *(problem in [20])* 3sum*: Given a set S of n integer numbers, are there $a, b, c \in S$ with $a + b + c = 0$?*

Definition 2. *(definition in [20]) A problem PR is* 3sum*-hard if* 3sum $\lll_{f(n)}$ *PR, where $f(n) = o(n^2)$.*

The fastest known algorithm for the 3sum problem runs in $\Theta(n^2)$ time.

Problem 2. *(problem in [20])* SegContPnt*(*segments containing points*): We are given a set P of n real numbers and a set Q of m pairwise-disjoint intervals of real numbers. Let $P = \{x_1, x_2, \cdots, x_n\}$ and $Q = \{[a_1, b_1], [a_2, b_2], \cdots, [a_n, b_n]\}$ in $\mathbb{R}$. Is there a real number (translation) u such that $P + u \subseteq Q$?*

SegContPnt is proven to be 3sum-hard in [20]. In other words, by the definition 2, any instance of 3sum can be solved by solving some instances of SegContPnt. (Remark: In [20], we can have $|P| < |Q|$, but since the reduction is made with an instance such that $|P| = |Q|$, the version above is also 3sum-hard.)

We will prove that the segment pattern matching problem below is also 3sum-hard. It is a decision problem, hence the corresponding optimization problem is also 3sum-hard.

Problem 3. MSLS *(*matching sets of line segments under translation with respect to the Hausdorff distance in 2D*): Given two sets of line segments S and S' in $\mathbb{R}^2$ such that $|S| = |S'| = n$, and given a tolerance $\delta > 0$, decide whether there exists a translation τ in $\mathbb{R}^2$ such that for all $s \in S$, there exists $s' \in S'$ satisfying $d(\tau + s, s') \leqslant \delta\ell(s)$.*

In order to prove that MSLS is 3sum-hard, it suffices to prove that SegCongPnt $\lll_{f(n)}$ MSLS, because the relation $\lll_{f(n)}$ is transitive. So given an instance (P, Q) of the SegContPnt problem, we construct the following instance of MSLS. We construct the line segments in S from the segments in Q and we construct the line segments in S' from the points in Q. (See Figure 13.) For all i and j, we set:

(i) $s_i = (p_i, q_i)$ where $\ell_i = \ell(s_i) = \frac{1+(b_i-a_i)^2}{2}$, and $p_i = (\frac{a_i+b_i}{2}, \frac{\ell_i}{2})$, $q_i = (\frac{a_i+b_i}{2}, -\frac{\ell_i}{2})$

(ii) $s'_j = (p'_j, q'_j)$ where $\ell'_j = 1$, and $p'_j = (x_j, \frac{1}{2})$, $q'_j = (x_j, -\frac{1}{2})$

(iii) $\delta = \frac{1}{\sqrt{2}}$

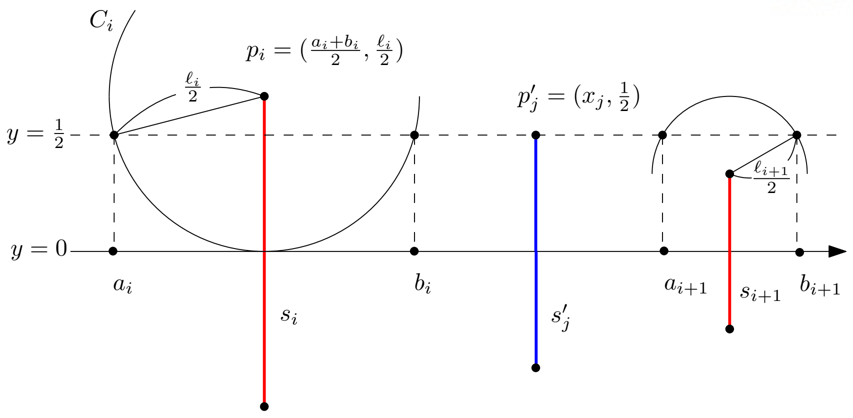


Figure 13: Reduction of SegContPnt to MSLS

Lemma 32. *Suppose that* $\tau = (u, v)$ *satisfies* $d(\tau + s, s') \leqslant \delta\ell(s)$. *Let* $\tau_h = (u, 0)$. *Then we have* $d(\tau_h + s, s') \leqslant \delta\ell(s)$.

Proof. Let $s \in S$ and $s' \in S'$. We define the function $f(y) = d^2((u, y) + s, s') - \delta^2\ell^2(s)$. This function is a polynomial of degree 2 in y. As the segments s and s' are symmetric about the x-axis, $f(y)$ is symmetric. Moreover, $f(y)$ goes to infinity when y goes to infinity, as the segment $(u, y) + s$ goes upwards to infinity. It follows that the minimum of $f(y)$ is achieved when $y = 0$. So we have $f(0) \leqslant f(v)$. Since $d(\tau + s, s') \leqslant \delta\ell(s)$, we have $f(v) \leqslant 0$. It follows that $f(0) \leqslant 0$, and thus $d(\tau_h + s, s') \leqslant \delta\ell(s)$ ☐

We now give a necessary and sufficient condition for two segments to match.

Lemma 33. *We have* $d((u, 0) + s_i, s'_j) \leqslant \delta\ell_i$ *if and only if* $u + x_j \in [a_i, b_j]$.

Proof. Let $g(u) = d^2((u, 0) + s_i, s'_j) - \ell_i^2$. We need to prove that $f(u) \leqslant 0$ if and only if $u + x_j \in [a_i, b_i]$. Let C_i denote the circle centered at p_i with radius $\ell_i/2$. The point $(u, 0) + p'_j$ is inside C_i if and only if $u + x_j \in [a_i, b_i]$. (See Figure 13.) This is true if and only if $d((u, 0) + p'_j, p_i) \leqslant \ell_i/2$. Symmetrically, it is equivalent to $d((u, 0) + q'_j, q_i) \leqslant \ell_i/2$. It means that $d^2((u, 0) + s_i, s'_j) \leqslant \ell_i^2/2$, and thus $d^2((u, 0) + s_i, s'_j) \leqslant \delta\ell_i$.

☐

Now we can prove the main result of this section.

Theorem 34. SegContPnt $\lll_{f(n)}$ MSLS, *where* $f(n) = o(n^2)$.

Proof. Suppose that our instance of SegContPnt is positive. In other words, there exists u such that for all j, there exists i such that $u + x_j \in [a_i, b_i]$. Then by Lemma 33, we have $d((u, 0) + s_i, s'_j) \leqslant \delta\ell_i$. In other words, our instance of MSLS is positive.

Conversely, suppose that our instance of MSLS is positive. Then there exists a translation $\tau = (u, v)$ such that for all i, there exists j such that $d(\tau + s_i, s'_j) \leqslant \delta\ell_i$. By Lemma 32, it implies that $d((u, 0) + s_i, s'_j) \leqslant \delta\ell_i$. Hence, by Lemma 33, we have $u + x_j \in [a_i, b_i]$. It means that our instance of SegContPnt is positive. ☐

Therefore our problem MSLS is 3SUM-hard, and a subquadratic algorithm for solving it is currently out of reach.

2.9 Future Work

Is there a characteristic in a set of line segments giving an advantage to match such as corner points by Efrat et al. [5]? Are the two characteristics, in a set of unit length of line segments and in a set of arbitrary length of line segments, same? These questions could be the future work.

criteria		
discrete	continuous or geometric setting	
centralized (coupled)	decentralized (decoupled)	
labeled	colored	unlabeled

Table 2: Criteria of our problem in bold.

III Multi-robot path planning

In this section, we present our results on path planning for multiple robots. Two preliminary versions of this section appeared in the proceedings of the *37th International Symposium on Computational Geometry [21]* and in the *Journal of Experimental Algorithmics [22]*.

3.1 Problem Statement

We first briefly describe this problem. A set of n robots, modeled as unit squares, need to move on the integer grid $\mathbb{Z}^2$, from their starting positions $s_1, \dots, s_n \in \mathbb{Z}^2$ to their target positions $d_1, \dots, d_n \in \mathbb{Z}^2$. A (possibly empty) set $\mathcal{O} \subset \mathbb{Z}^2$ of stationary obstacles is also given. We denote by $p_i(t) = (x_i(t), y_i(t)) \in \mathbb{Z}^2$ the position of robot i at time $t \in \mathbb{N}$. At each time $t \in \mathbb{N}$, the robot may either stay at the same position, or move to one of the four neighboring squares; hence we have $p_i(t+1) - p_i(t) \in \{(0,0), (-1,0), (0,-1), (0,1), (1,0)\}$.

While moving, each robot must avoid collision with obstacles and other robots. In particular, for any robot i and any time $t \in \mathbb{N}$, we must have $p_i(t) \notin \mathcal{O}$ and $p_i(t) \neq p_j(t)$ for all $j \neq i$. In addition, a constraint is enforced in order to model coordinated movement: a robot i can only move at time t to the position previously occupied at time $t-1$ by another robot j if they move in the same direction. More precisely, if $p_i(t+1) = p_j(t)$, then we must have $p_i(t+1) - p_i(t) = p_j(t+1) - p_j(t)$.

The *length* of the trajectory of robot i is the number of times robot i moves; in other words it is $|\{t \in \mathbb{N} \mid p_i(t+1) \neq p_i(t)\}|$. Its *completion time* $t_C(i)$ is the time when robot i ceases to move; in other words, it is the minimum time $t_C(i)$ such that for all $t \geqslant t_C(i)$, we have $p_i(t) = d_i$. Even if a robot reaches its target position, it does not disappear from the grid. The *makespan* is the maximum completion time among all robots. The problem we will solve looks like Figure 3. We assume that three robots numbered 1 to 3 will move from the red pixel (start position) to the blue pixel (target position). They move at unit speed without collision. In this environment and conditions, there are two objectives. First we may want to reduce the completion time. In other words, the time when last robot reaches its target position is minimized. Or we may want to minimize the total lengths of the robots paths. Therefore, we design optimization algorithms for the different goals, i.e. cost functions. Table 2 presents the criteria for coordinating multi-robot path planning.

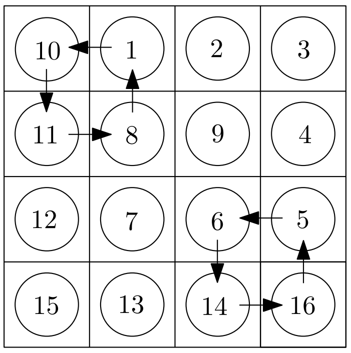


Figure 14: Multi-disk shaped objects parallel motion planning on a grid.

3.2 Related Work

Our path planning problem is a relaxed version of the 15-puzzle, but is more restricted than pebble motion. (See Figure 4 and Figure 14.) The 15-puzzle problem allows one empty pixel and our problem is on the unbounded grid. Ratner and Warmuth [23] proved that finding a shortest solution of 15-puzzle is NP-hard. Efficient algorithms for deciding reachability of a target configuration were given by Wilson [24] and Kornhauser, Miller, and Spirakis [25]. Pebble motion, or swarm robots in [26] has a mild assumption of separation between robots, so it allows some directed cycles. On the other hand, the problem studied in this section does not allow directed cycles. In order to achieve the same configuration after one move in such a cycle, our problem needs at least three extra steps.

This coordinated path planning problem was the third computational geometry challenge [26, 27]. As we mentioned, the solutions to this coordinated path planning problem were scored according to two criteria: we should either minimize the makespan (MAX), or minimize the sum of the lengths of the paths (SUM). Our team (UNIST) ranked second according to both criteria, out of 17 teams participating in the contest. We obtained the highest score in 120 SUM instances out of 203. Team Shadoks [28] ranked first according to MAX and third according to SUM, while team Gitastrophe [29] ranked first according to SUM and third according to MAX. The other two teams also start from computing the feasible solution (but in the different way) and then optimize it. Shadoks had 202 best solutions out of 203 instances by using their *conflict optimizer* which performs very well in the MAX version. Gitastrophe used the *k-opt* method which chooses k robots and improve their paths while other robots' paths are fixed. In a recent paper [29], the Gitastrophe team applied the Shadoks' confilct optimizer to their k-opt method. More information on the contest can be found in the survey by Fekete et al. [27].

3.3 Methods

In this section, we present our three algorithms: the algorithm for computing an initial feasible solution (F), our simple local search algorithm (LS) and our simulated annealing approach (SA). We begin with the description of our data structure, which is common to these three algorithms.

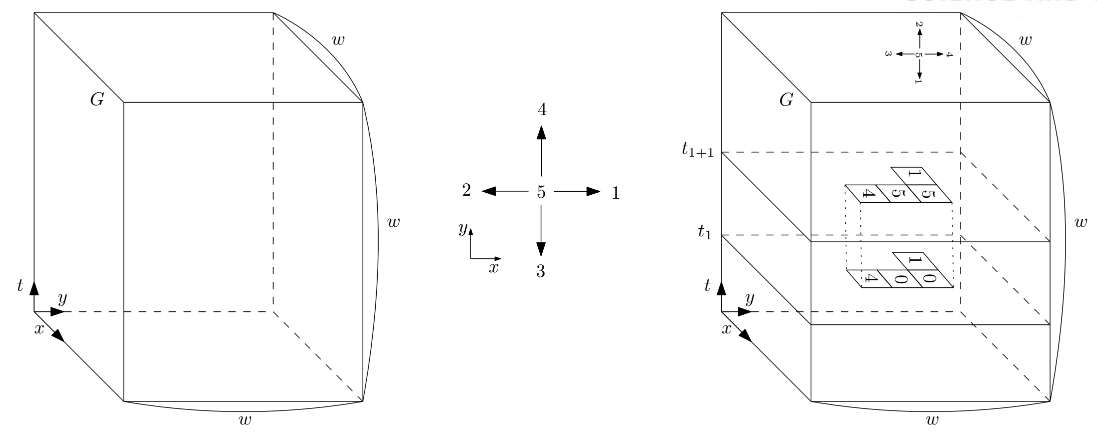


Figure 15: Our data structure G.

3.3.1 Data structure

Our data structure consists of two 3-dimensional arrays G and H. The array G is a 3D-array of 8-bit integers, where $G[x, y, t] = -1$ if (x, y) is an obstacle, $G[x, y, t] = 0$ if the cell (x, y) is empty at time t, and if $(x, y) = p_i(t)$ for some i, then $G[x, y, t]$ records $p_i(t + 1) - p_i(t)$. In other words, if robot i is located at (x, y) at time t, then $G[x, y, t]$ records the direction taken by robot i. We encode this direction as an integer in $\{1, \dots, 5\}$, where 5 means that robot i does not move. (In particular, G and H do not record robot numbers.) The array H is a 3D-array of 16-bit integers, which is only needed in the SUM version.

The size of the array is chosen as follows. In the x and y directions, the array corresponds to the minimum bounding box of the middle points as shown in Figure 17. (See explanation in Section 3.3.2.) In the time direction, we expand the array as needed by our algorithm when it constructs the solution.

For the sake of analyzing our algorithms, we use w to denote the size of G in either dimension. The reason is that, for the problem instances given in the contest, the makespan of the solutions we constructed was not much larger than the length and width of the array we used—less than a factor 10. So we may assume that G is a $w \times w \times w$-array. Our data structure allows us to do the following.

Deletion. We can delete the trajectory of robot i from G in $O(w)$ time. It suffices to follow the direction given by $G[x, y, t]$ from the starting point s_i.

Insertion. Assuming robot i is not yet recorded in G, and there is a feasible path from s_i to d_i, we can insert in $O(w^3)$ time a feasible trajectory of robot i that either minimizes the completion time $t_C(i)$, or minimizes its length. It can be done by a simple sweep of G by increasing values of t, and storing in $G[x, y, t]$ the direction of a move (if any) leading to (x, y, t) as an integer in $\{11, \dots, 15\}$ (in order to be able to reconstruct a feasible path backwards). In the SUM version, we also record in $H[x, y, t]$ the length of the shortest feasible trajectory to (x, y, t), which allows

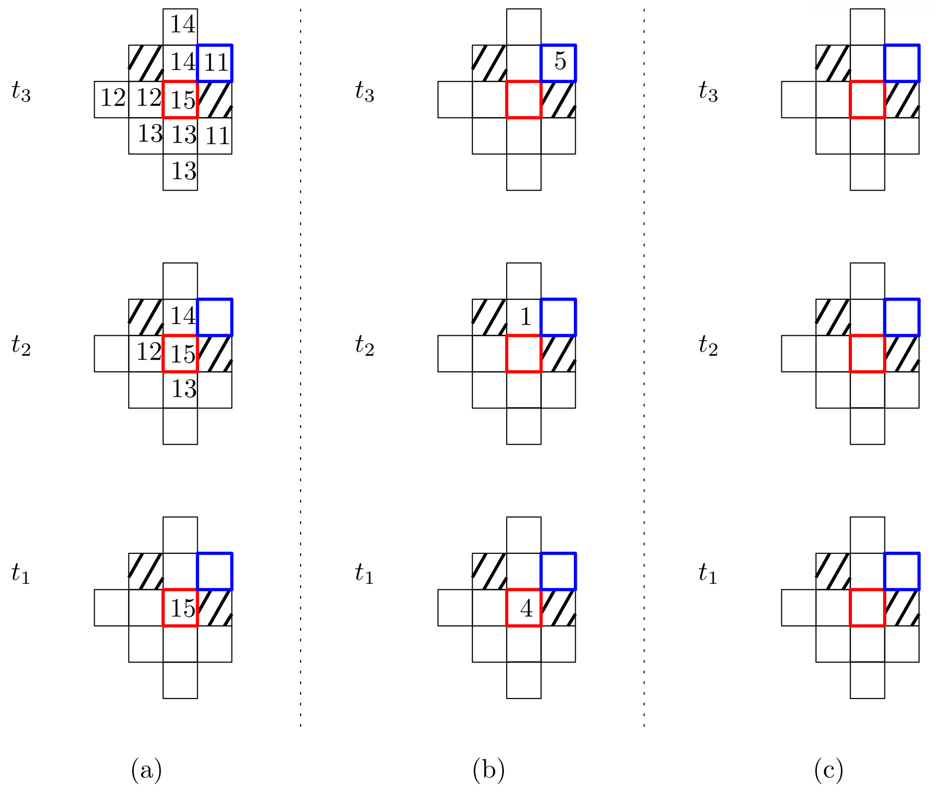


Figure 16: (a)-(b) is the insertion operation in $O(w^3)$ and (b)-(c) is the deletion operation in $O(w)$.

us to return a feasible path with minimum length. At the end of the sweep, we reconstruct the path backwards, and we remove all the values larger than 10 from G.

A detailed pseudocode for this insertion operation in the MAX version is given as Algorithm 1. The corresponding C++ code is the function `UNISTCG21_LS::MoveOneRobot` in the file `ls.cpp` at: https://github.com/antoinevigneron/CGSHOP21/tree/main/LocalSearch

The first phase, where we sweep the array from $t = 0$ to $t = w - 1$ and encode the reachable configurations with integers in $\{11, \ldots, 15\}$, appears between lines 3 and 25. During this phase, a configuration $(x_2, y_2, t + 1)$ may be reached from several directions. In this case, we record in $G[x_2, y_2, t+1]$ only one direction leading to $(x_2, y_2, t+1)$ and we give priority to direction 5, which means that the robot does not move between time t and $t + 1$ (Lines 4– 7). It guarantees that, when we reconstruct the path backwards, we obtain the feasible path with minimum completion time.

Note that this temporary encoding of directions for robot i with integers in $\{11, \ldots, 15\}$ is different from the encoding of the other robots $j \neq i$ that are recorded in G: For such a robot j, the direction is encoded by an integer in $\{1, \ldots, 5\}$, and $G[x, y, t]$ encodes the direction robot j takes *from* (x, y, t) instead of the direction *to* (x, y, t).

Our algorithm needs to take into account the constraint on coordinated movement stated in Section I: if $p_i(t + 1) = p_j(t)$, then we must have $p_i(t + 1) - p_i(t) = p_j(t + 1) - p_j(t)$. This is handled by two conditions named *swarm condition* 1 and 2 in the pseudocode (Lines 17–24).

Algorithm 1 Inserting a robot into our data structure.

```
1: procedure INSERT(robot $i$)
2:     $G[x(s_i), y(s_i), 0] \leftarrow 15$ ▷ assume that robots do not move between time $t = -1$ and $t = 0$
3:     for $t \leftarrow 0, w-2$ do    ▷ compute reachable configurations by sweeping the whole array
4:         for $(x, y) \in \{0, \ldots, w-1\}^2$ do    ▷ robot $i$ does not move between time $t$ and $t+1$
5:             $g \leftarrow G[x, y, t]$, $g_2 \leftarrow G[x, y, t+1]$
6:             if $g \geqslant 11$ and $g_2 = 0$ then    ▷ robot $i$ can reach $(x, y, t+1)$ coming from $(x, y, t)$
7:                 $G[x, y, t+1] \leftarrow 15$
8:         for $(x, y) \in \{0, \ldots, w-1\}^2$ do    ▷ robot $i$ does move between time $t$ and $t+1$
9:             $g \leftarrow G[x, y, t]$
10:            if $g \leqslant 5$ then
11:                continue    ▷ robot $i$ cannot reach $(x, y, t)$
12:            for $d \in \{1, 2, 3, 4\}$ do
13:                $(x_2, y_2) \leftarrow$ Move$(x, y, d)$    ▷ position obtained by moving in direction $d$ from $(x, y)$
14:                $g_2 \leftarrow G[x_2, y_2, t+1]$
15:                if $g_2 \neq 0$ then
16:                    continue    ▷ target position is not free
17:                $g_3 \leftarrow G[x_2, y_2, t]$
18:                if $g_3 \in \{1, 2, 3, 4, 5\}$and $g_3 \neq d$ then    ▷ swarm condition 1
19:                    continue
20:                $g_4 \leftarrow G[x, y, t+1]$
21:                if $g_4 \in \{1, 2, 3, 4, 5\}$ then    ▷ swarm condition 2
22:                    $(x_3, y_3) \leftarrow$ Move$(x, y, -d)$ ▷ moving from $(x, y)$ in direction opposite from $d$
23:                    if $G[x_3, y_3, t] \neq d$ then
24:                        continue
25:                $G[x_2, y_2, t+1] \leftarrow d + 10$
26:     $(x, y) \leftarrow d_i$
27:     for $t \leftarrow w-1, 0$ do    ▷ reconstruct the trajectory of robot $i$ backwards from $d_i$
28:         $d \leftarrow G[x, y, t] - 10$
29:         if $t = w-1$ then
30:             $G[x, y, t] \leftarrow 5$ ▷ assume that robots do not move between time $t = w-1$ and $t = w$
31:         else
32:             $G[x, y, t] \leftarrow d'$    ▷ robot $r_i$ comes from direction opposite from $d'$
33:         $(x, y) \leftarrow$ Move$(x, y, -d)$
34:         $d' \leftarrow d$
35:     remove from $G[]$ all values $\geqslant 11$
```

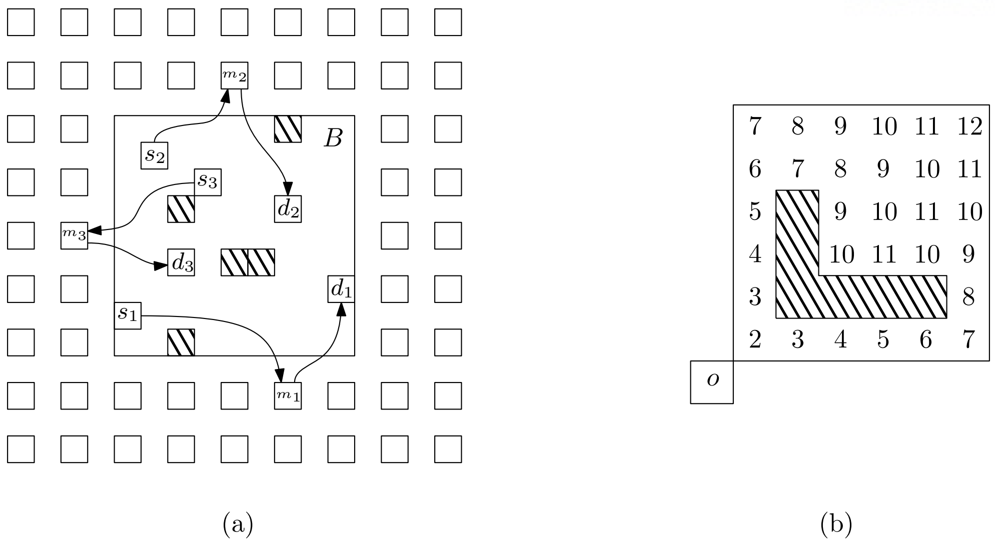


Figure 17: (a) Robot i goes from s_i to d_i through m_i. (b) The geodesic distance from o.

3.3.2 Computing a feasible solution

In order to compute a feasible solution, we construct a middle-point m_i for each robot i, such that there is a feasible path from s_i to d_i through m_i. (See Figure 17a.) Let B be the minimum bounding box of the starting points, target points and obstacles. The middle-points are chosen from the set of grid points outside B, not adjacent to B, and whose x and y-coordinates are even.

In a first stage, we move all the robots to their middle-points, and in a second stage we move them from their middle-points to their target points. In order to do this, we insert the robots one by one in our 3D array G, by applying the insertion procedure from Section 3.3.1. For the MAX version of the problem, we apply the version of the insertion algorithm that minimizes the makespan of the inserted robot, and for the SUM version we use the version that minimizes the length.

One difficulty here is that a robot can be blocked by other robots that are still at their starting position, so we need to move the robots in an appropriate order. To this end, we compute the geodesic distances from an arbitrary point o outside B to all the starting positions. (The geodesic distance between two cells of the grid is the length of the shortest obstacle-avoiding path between them, see Figure 17b.) We sort the robots by *increasing* geodesic distance, and insert them in this order. This ensures that there is always a feasible path from s_i to m_i for each i. In the second stage, we proceed in the same way, except that we proceed by *decreasing* geodesic distance from o to the target points. This approach finds a feasible solution to all the input instances of the contest, as all the robots are in the unbounded face of $\mathbb{Z}^2 \setminus \mathcal{O}$.

For each robot i, we have several choices for the middle-point m_i. We tried a few possibilities. The one that most often gives the best results was to choose the available middle-point that minimizes the sum of the Manhattan distances to s_i and d_i.

For each robot, we run the insertion procedure twice: once from the starting point, and once from the middle-point. Each insertion takes $O(w^3)$ as we sweep the whole grid. So we obtain a feasible solution in $O(nw^3)$ time.

3.3.3 Simple local search

In order to improve the feasible solution from Section 3.3.2, we can simply pick a robot i, delete it from G and insert it again using the procedures from Section 3.3.1. It may give a large improvement for this robot, as it no longer has to go through its middle-point m_i. We call this operation a *tightening* move. (See Figure 18.)

Our first optimization algorithm repeatedly tightens the path of a robot chosen at random. (See Algorithm 2.) We first compute a random permutation of the robots. Then we go through this random list, and perform a tightening operation on the current robot. If at least one of the trajectories was shortened, then we repeat this process with a new random permutation. (In the MAX version, we consider that the trajectory was shortened if its completion time is smaller, and in the SUM version we consider that it was shortened if its length is smaller.) Otherwise, as no progress was made, we restart from the input feasible solution (ignoring all intermediate improvements).

Algorithm 2 Our first local search algorithm. At Line 9, the new path of r_i is of minimum length for the SUM version, and minimum completion time for the MAX version.

```
 1: procedure SimpleLocalSearch
 2:     while true do
 3:         IMPROVED ← true
 4:         while IMPROVED = true do
 5:             IMPROVED ← false
 6:             compute a random permutation $(r_1, \ldots, r_n)$ of the robots.
 7:             for $i \leftarrow 1, n$ do
 8:                 delete the trajectory of $r_i$ from the 3D grid
 9:                 insert $r_i$ in the 3D grid along an optimal path
10:                 if the trajectory of $r_i$ was shortened then
11:                     IMPROVED ← true
12:         restart from the input feasible solution
```

3.3.4 Simulated annealing

One drawback of the local search approach described in Section 3.3.3 is that the only type of moves that is allowed consists in shortening the whole trajectory of one robot, and thus it can easily be trapped in a local minimum. To remedy this, we use a simulated annealing approach [30] that makes two types of moves: tightening moves as described in Section 3.3.3, and *stretching* moves, which can make a trajectory longer.

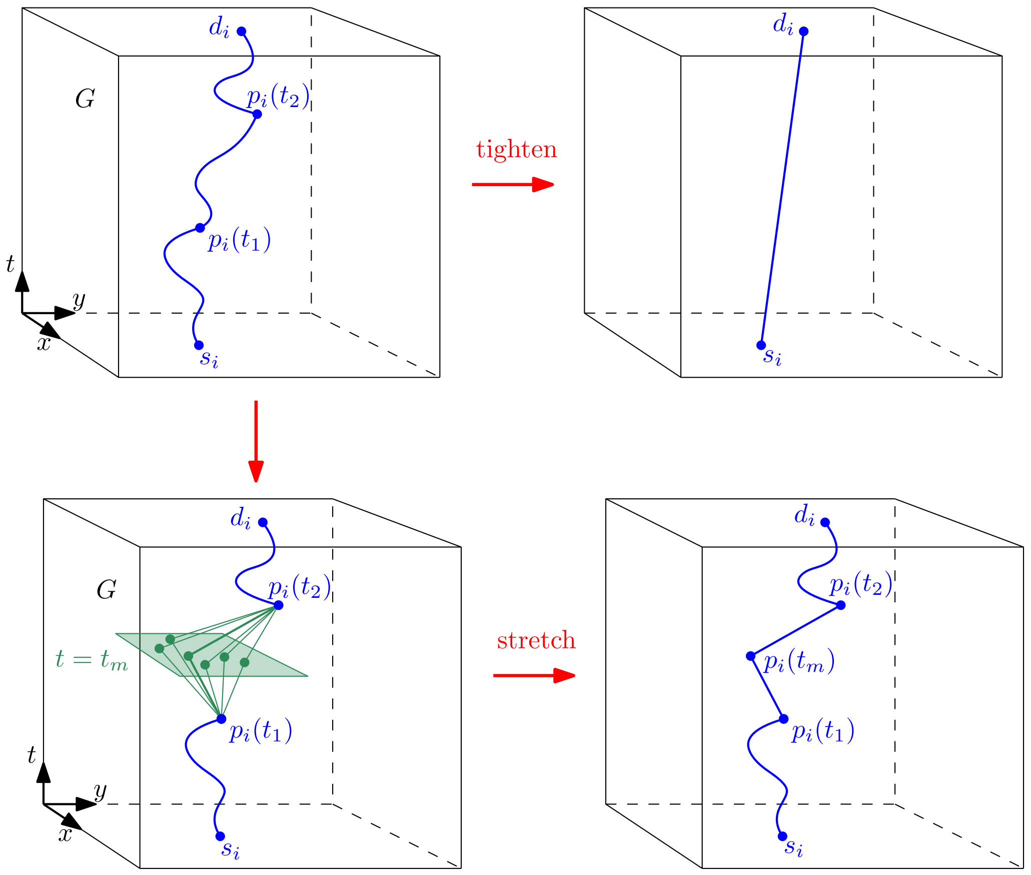


Figure 18: Tightening and stretching moves.

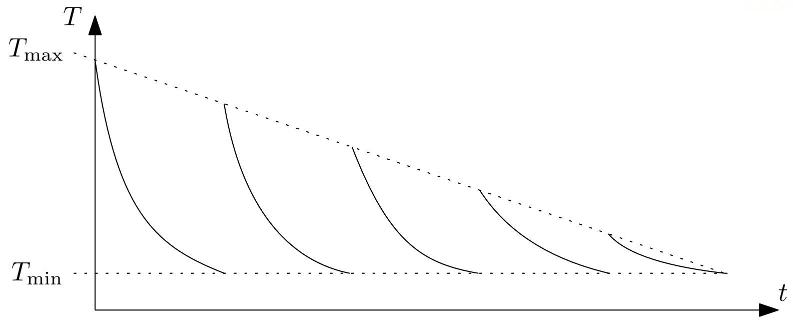


Figure 19: A cooling schedule with $n_{\text{cycle}} = 5$.

We now describe stretching moves. (See Figure 18.) As mentioned above, we assume that G is a $w \times w \times w$ array. Let i be a robot, and let t_1, t_2 be integers such that $0 \leqslant t_1 \leqslant t_2 < w$. We will see later how we generate t_1 and t_2 at random. We first delete the trajectory of i between $p_i(t_1)$ and $p_i(t_2)$. Let t_m be chosen uniformly at random between t_1 and t_2. We compute all of the cells (x, y, t_m) of G that are reachable from $p_i(t_1)$, and that are reachable backwards from $p_i(t_2)$. We pick one of these points uniformly at random, which we denote by q. Then we connect q to $p_i(t_1)$ and $p_i(t_2)$ through shortest paths, computed in the same way as we did in the insertion operation from Section 3.3.1.

We now explain how we generate t_1 and t_2. We first set $\delta = \min(w-1, \lfloor \alpha\sqrt{1/x} \rfloor)$, where α is a constant larger than 2, and x is a random floating point number in $(0, 1]$. Then we generate t_1 as a random number chosen uniformly between 0 and $w - 1 - \delta$, and we set $t_2 = t_1 + \delta$. This approach ensures that our stretching operation takes $O(w)$ expected time. (See Theorem 35.)

In the SUM version, the score we attempt to minimize is the sum of the lengths of the paths divided by the number of robots. In the MAX version, the score is MAX$+(n_{\max} - 1)/n$, where MAX is the makespan, and $n_{\max}$ is the number of robots whose completion time is equal to MAX. The reason for introducing the second term is that otherwise, the algorithm may worsen the solution by increasing the completion time of some robots to match the makespan, without it being reflected in the score. In our experiments, we observed that this second term improves the results.

A move that improves the score is always accepted by the algorithm, but a move that increases it is accepted with probability $\exp(-\Delta/T)$, where T is the current temperature and Δ is the score increase. We used a cooling schedule consisting of n_{cycles} cycles of n_{iter} iterations each, such that the temperature decreases exponentially within each cycle, and the maximum temperature decreases linearly. (See Figure 19.)

3.4 Algorithm Engineering

In this section, we analyze our algorithm, and discuss issues related to the tuning of our simulated annealing approach.

3.4.1 Running time analysis

As mentioned above, our algorithm for computing an initial solution (F) runs in $O(nw^3)$ time, and our simple local search algorithm (LS) runs in $O(w^3)$ time per iteration. We now analyze our simulated annealing algorithm.

Theorem 35. *Our simulated annealing algorithm (SA) runs in expected time $O(w^3)$ per tightening move and $O(w)$ per stretching move.*

Proof. As the tightening operation consists in deleting and inserting a path in our 3D grid, it takes $O(w^3)$ time. We now consider a stretching move on robot r_i. We only update cells $G[x, y, t]$ of the grid such that $t_1 \leqslant t \leqslant t_2$.

Remember that $\delta = t_2 - t_1$. We denote by (x_1, y_1) the coordinates of r_i at time t_1. The x and y-coordinate increase or decrease by at most 1 when t increases by 1, hence we only traverse cells (x, y, t) of the grid such that $x_1 - \delta \leqslant x \leqslant x_1 + \delta$ and $y_1 - \delta \leqslant y \leqslant y_1 + \delta$. So during a tightening move, we only update a sub-array of size at most $(2\delta + 1) \times (2\delta + 1) \times (\delta + 1)$, and thus we can sweep this sub-array in $O(\delta^3)$ time. In addition, we need to compute $p_i(t_1)$, which takes $O(w)$ time by reconstructing the path of r_i. Thus a stretching operation takes time $O(w + \delta^3)$. We now bound the expected value $E[\delta^3]$ of δ^3.

We have $\delta = \min(w - 1, \lfloor \alpha\sqrt{1/x} \rfloor)$ where x is a random number in $(0, 1]$ and $2 \leqslant \alpha = O(1)$. So for $2 \leqslant k \leqslant w - 1$, we have

$$\begin{aligned}\Pr[\delta = k] &= \Pr\left[k \leqslant \alpha\sqrt{1/x} < k + 1\right] = \Pr\left[k^2 \leqslant \alpha^2/x < (k + 1)^2\right] = \Pr\left[\alpha^2/(k + 1)^2 < x \leqslant \alpha^2/k^2\right] \\ &= \alpha^2\left(\frac{1}{k^2} - \frac{1}{(k + 1)^2}\right) = \alpha^2\frac{2k + 1}{k^2(k + 1)^2} \leqslant \frac{2\alpha^2}{k^3},\end{aligned}$$

which implies that

$$\begin{aligned}E[\delta^3] &= \left(\sum_{k=2}^{w-2} \Pr[\delta = k]k^3\right) + \Pr[\delta = w - 1](w - 1)^3 \\ &\leqslant 2\alpha^2(w - 1) + \Pr[x \leqslant \alpha^2/w^2](w - 1)^3 \\ &= 2\alpha^2(w - 1) + \frac{\alpha^2}{w^2}(w - 1)^3 \leqslant 3\alpha^2 w = O(w)\end{aligned}$$

and the result follows. □

3.4.2 Space usage

For our three algorithms (F, LS and SA), the size of the data structure (Section 3.3.1) is dominated by the arrays G and H, which consist of w^3 8-bit and 16-bit integers, respectively. So in the MAX version, the size is roughly w^3 bytes and in the SUM version, it is $3w^3$ bytes. We chose this data structure in order to minimize memory usage, as we feared that large instances would not fit in RAM otherwise. (Our server has 4 Intel Xeon E5-2697 V4, 2.3 GHz, 72 cores in total under Linux. The size of the RAM is 251 Gigabytes.)

It turns out that we only use a small percentage of our RAM even for the largest instances (less than 7%), so we could have afforded a larger data structure. On the other hand, it is unclear how it could have been useful with our approach. For instance, we could have recorded the trajectory of each robot separately. When we run a stretching operation, it allows us to find in constant time the first point $p_i(t_1)$ along which we stretch, and bring down the running time of a stretching operation from $\Theta(w + \delta^3)$ to $\Theta(\delta^3)$.

We made an experiment to check whether this could be helpful. So at each execution of a stretch operation, we locate the point $p_i(t_1)$ twice, instead of once, using our current procedure that takes $\Theta(w)$ time. For a small instance (`small_004`), it makes the algorithm 6% slower, and for a large instance (`london_night_00009`), it makes it 0.6% slower. So the improvement we could obtain by recording explicitly all the trajectories would be modest.

3.4.3 Parameters of our simulated annealing approach

One advantage of our simulated annealing approach is its flexibility: We can change the objective function (SUM or MAX), and we can adjust parameters such as the maximum and minimum temperatures, the frequency of the different types of moves, and we can modify the cooling cycle. A good choice of parameters yields markedly improved results. On the other hand, as we shall see below, it comes with the drawback that it is not easy to find such a good set of parameters.

The main parameters are the following: The number n_{cycles} of cooling cycles, the number n_{iter} of iterations per cooling cycle, the maximum temperatures $T_{\max}$, and the frequencies f_s and f_t of the stretching and tightening moves. More precisely, the probability of attempting a stretching move or a tightening move are $f_s/(f_s + f_t)$ and $f_t/(f_s + f_t)$, respectively.

Our SA program takes as input an optional parameter file that allows the user to modify these parameters. A few other parameters are included, that seem to have less impact on the result, in the sense that we found values for these parameters that appear to work well for all instances. Two of these parameters specify how much margin we allow in the grid around the minimum bounding box of the input solution. The parameter α, described in Section 3.3.4 determines the length of a stretching move. The minimum temperature is $T_{\min}$. We observed, for instance, that $\alpha = 5$ gives good results, as well as $T_{\min} = 10^{-4}$.

We determined the values of n_{cycles}, n_{iter}, $T_{\max}$, f_s, f_t by trying on several instances of various sizes, varying the parameters and comparing the results obtained. Since f_s and f_t are not independent—more precisely we can replace them with 1 and f_t/f_s without changing the behavior of our algorithm—there are 4 degrees of freedom in our search. In addition, in order to get non-trivial results for large instances, each experiment takes hours or days. So we may not have found the best set of parameters. Table 4 shows the parameters we used for the experiments presented in Section 3.5.

To be more specific, we now show on a small example how we tuned these parameters. We

	A1	A2	**A3**	A4	A5	B1	B2	**B3**	B4=A3	B5
$T_{\max}$	0.1					10^{-4}	10^{-3}	$\mathbf{10^{-2}}$	0.1	1
n_{iter}	10^5					10^5				
n_{cycles}	10^2					10^2				
f_s	1	1	**10**	10^2	10^3	10				
f_t	10	1	**1**	1	1	1				
time (s)	5876	2316	**615**	622	297	621	666	**594**	620	873
MAX	75	61	**57**	147	162	56	55	**52**	57	54

Table 3: Tuning the parameters of our simulated annealing algorithm. In all cases, we use $T_{\min} = 10^{-4}$ and $\alpha = 5$. We start from a solution to instance `small_017` with makespan 162.

ran calculations on the instance `small_017`, which has 322 robots. They lasted about 10 minutes (see Table 3) on a workstation (AMD Ryzen 3500x). We optimized the makespan, i.e. we used the MAX version. We started from a feasible solution with makespan 162.

We first tried to determine the relative frequencies of stretching and tightening moves. These experiments are labeled A1–A5 and B1–B5 in Table 3. So we varied f_s and f_t, keeping the other parameters fixed. For instance, in column A1, $f_s = 1$ and $f_t = 10$ so tightening moves are 10 times more frequent than stretching moves. As tightening moves take more time than stretching moves, the times taken for these experiments vary substantially: A1 takes about 100 minutes and A5 takes only 5 minutes. The best results are obtained in experiment A3, with a makespan of 57. A5 completes about twice as fast, but gives a much worse result (makespan 162).

Then we started from the parameters of experiment A3, and varied the maximum temperature $T_{\max}$. So we have a new set of experiments B1–B5, were B4=A3. The times taken still differ, but by a smaller amount than it was in experiments A1–A5. The reason is that the time taken for each move is proportional to the makespan of the current solution, so a set of parameters that leads to finding better solutions earlier will lead to a shorter running time. The best solution in experiments B1–B5 is given by experiment B3, with a maximum temperature $T_{\max} = 10^{-2}$. We obtain worse solutions for lower and higher temperatures.

We just showed two steps in the process of tuning the parameters of our algorithm. In order to obtain the full set of parameters, we iterated this process several times, using different parameters. For instance, after experiments B1–B5, we could try to find a better trade-off between the number of iterations and the number of cycles. So we could use $n_{\text{iter}} = 10^4$ and $n_{\text{cycles}} = 10^3$, or $n_{\text{iter}} = 10^6$ and $n_{\text{cycles}} = 10$. We may need several more iterations to converge to a suitable set of parameters.

3.4.4 Failed attempts

During the competition, we tried other sets of moves than the ones described above. In particular, we tried to do tightening moves in a time window $[t_1, t_2]$, as we do for stretching moves, instead

	MAX			SUM		
	small	medium	large	small	medium	large
$T_{\max}$	0.1	0.1	0.1	10^{-4}	10^{-4}	10^{-4}
n_{iter}	10^7	10^5	10^6	10^8	10^8	10^4
n_{cycles}	10^3	10^3	10	10^3	10^3	10^3
f_s	10	1	1	5.10^3	10^3	1
f_t	1	1	10^3	1	1	1

Table 4: Simulated annealing parameters for our experiment. In all cases, we use $T_{\min} = 10^{-4}$ and $\alpha = 5$.

of $[0, w]$. We also tried to use tightening moves or stretching moves that also minimize the sum of the lengths in the MAX version. These approaches made the results worse.

We also tried to use a smoothed objective function, using a linear combination of the makespan and the sum of the lengths. The idea was that the makespan does not change smoothly, and especially for large instances, it may not make any progress for long periods of time. This approach also made the results worse.

3.5 Results

We implemented the algorithms above in C++. The source code can be found at:

github.com/antoinevigneron/CGSHOP21.

We denote by F, LS and SA the algorithm that generates the initial feasible solution (Section 3.3.2), the simple local search algorithm (Section 3.3.3) and the simulated annealing algorithm (Section 3.3.4), respectively.

During the contest, we first had F available, then LS, and finally SA. We are able to find a feasible solution for any particular instance of the contest in less than 2 hours. In particular, we ran SA on the solutions produced by LS, but we did not keep track of all the running times. In order to show a fairer comparison, we present new experiments in this paper. We choose 24 instances of various sizes, and compute an initial feasible solution for each instance using F, which has two versions: one for SUM and one for MAX. Then we run LS and SA on each of these feasible solutions. Figure 20 shows the results on these 24 instances grouped into small, medium and large size, after running either LS for 2 days or SA for 2 days. Table 5 and Table 6 record the related data.

3.5.1 Parameters

The parameters we use are shown in Table 4. There are different sets of parameters for each group of 8 solutions, and for each version MAX or SUM. We determined these parameters by hand, as explained in Section 3.5. The corresponding parameter files are provided with our source code, in the subdirectory `210905Parameters`.

For the MAX version, we adjusted the number of cycles n_{cycles} and the number of iterations per cycle n_{iter} so that the computation runs over 2 days, and completes a reasonable number of cycles. This could not be perfectly adjusted as the sizes vary. We use a larger number of stretching moves for small instances.

In the SUM version, we ended up setting the temperature to 10^{-4}, and remaining constant during the whole course of the algorithm. At this temperature, all the moves that increase the objective function are rejected, so we are effectively doing local search, but with a different set of moves from what we used in Algorithm 2, the simple local search approach. More precisely, we are now allowing stretching moves on a robot that do not increase the length of its path. This can happen for two reasons: First there could be a local change that keeps the number of moves the same, or a move by another robot might have given more free space for the current robot, allowing to decrease the length of the trajectory.

A tightening move takes $O(w^3)$ time, while a stretching move takes $O(w)$ expected time. (See Section 3.4.1.) So we were expecting it to be advantageous to make a larger number of stretching moves. However, our experiments show that it does not always help. In particular, for large instances in the SUM version, the stretching and tightening moves have the same frequency, and in the MAX version tightening moves are 10^3 times more frequent. (See Table 4.) Our efficient implementation of the stretching operation is still helpful for small instances, where we make stretching moves 10 and 5000 times more frequent than tightening moves in the MAX and SUM version, respectively.

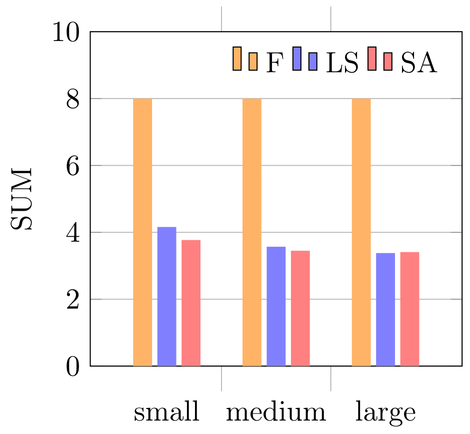
3.5.2 Discussion

Figure 20 shows that in the MAX version, SA is substantially better than LS for small instances, and LS outperforms SA for medium and large-sized instances by a small margin. However, for medium-size instances, Table 5 shows that it is only due to the largest instance in this category, for which the result of LS is almost twice as good as the result of SA. The reason is that the set of parameters we use for medium-size instances is not adapted for this particular instance. Our set of parameters for large instances works better in this case. It illustrates the difficulty of choosing a good set of parameters for some instances.

In the SUM version, our simulated annealing approach (SA) operates as local search with stretching and tightening moves (see Section 3.5.1). It outperforms LS on small and medium instances, and LS does slightly better on large instances.

The main advantage of our simple local search algorithm (LS), which only uses tightening moves, is that it runs without any parameter, and gives reasonably good solutions in most cases. It was especially helpful during the competition as we could start running it earlier than SA without having to spend time tuning it.

Figure 21 and Figure 22 show the score over time when we run LS and SA on two instances,



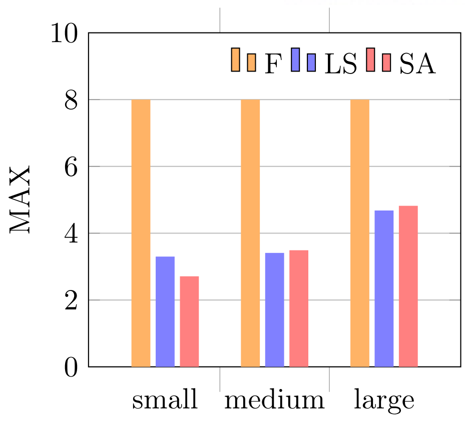


Figure 20: Performance of our algorithms on 8 small, 8 medium and 8 large-size instances. Results are normalized according to the score for F, the initial feasible solution. Our simple local search algorithm (LS) and simulated annealing (SA) were run for 48 hours each.

`clouds_00003` (1000 robots) and `medium_018` (1993 robots). We can see that LS improves the score faster in the beginning, and then SA catches up. Our interpretation is that LS tries to improve the score more aggressively using tightening moves, which leads more quickly to a local minimum. On the other hand, SA does tightening moves on short portions of the trajectories which provide smaller improvements in the beginning, but allow the algorithm to explore a larger portion of the search space.

3.6 Future Work

Our algorithm for computing an initial feasible solution (F) can be improved: In particular, the approaches by teams Gitastrophe and Shadoks appear to be much better [28, 29]. One question is whether starting from better initial solutions would lead to better solutions after simulated annealing (SA), or whether SA would close the gap after a reasonable time? Or perhaps it would help to use an algorithm that produces a large variety of initial solutions, and then to apply SA on them? It might be especially important for large instances where SA may not be able to explore a large enough portion of the feasible space.

Another issue is the choice of parameters for simulated annealing. The parameters we use in this paper give better results than what we used during the contest, but there could be room for improvement. The difficulty is that for large instances, we may not see any substantial improvement in the scores within a whole day of computation, so the process of improving the parameter set takes time and computing resources, whether we do it by hand or using another optimization algorithm. In addition, our experiments seem to show that some instances of similar sizes may require different parameter sets, so it is unclear how to predict which parameters would be suitable for which solution. A better approach might thus be to allow the algorithm to change the parameters during the computation. For instance, after detecting that the score is increasing too much, the algorithm could decrease the temperature or the frequency of stretching moves.

instance name	nb. of robots	normalized makespan SA	normalized makespan LS	normalized makespan F	makespan F
small instances					
buffalo_free_000_125	125	0.38	0.39	1	88
galaxy_cluster2_00002_251	251	0.26	0.37	1	161
algae_00001_278	278	0.27	0.40	1	142
small_free_017_320	320	0.28	0.40	1	152
medium_free_003_320	320	0.44	0.45	1	143
small_018_324	324	0.27	0.39	1	179
medium_005_407	407	0.38	0.43	1	245
sun_00001_571	571	0.43	0.47	1	256
sum		2.71	3.30	8	
medium instances					
clouds_00003_1000	1000	0.33	0.37	1	258
algae_00004_1113	1113	0.36	0.39	1	314
medium_017_1202	1202	0.58	0.64	1	481
buffalo_004_1404	1404	0.32	0.34	1	355
large_001_1563	1563	0.36	0.37	1	375
sun_00004_1707	1707	0.38	0.41	1	418
medium_018_1993	1993	0.45	0.51	1	506
microbes_00006_2500	2500	0.71	0.38	1	470
sum		3.49	3.41	8	
large instances					
universe_bgradiation_00006_3000	3000	0.41	0.40	1	503
algae_00007_4000	4000	0.39	0.38	1	548
redblue_00009_4500	4500	0.39	0.38	1	600
galaxy_cluster_00008_5000	5000	0.38	0.36	1	726
london_night_00009_6000	6000	0.36	0.38	1	812
clouds_00009_7229	7229	0.94	0.86	1	937
universe_bgradiation_00009_8000	8000	0.96	0.94	1	968
large_free_009_9000	9000	0.99	0.98	1	1134
sum		4.82	4.68	8	

Table 5: A few instances with the scores obtained after running our algorithms for 48 hours.

instance	nb. of	normalized sum of length			sum len.
name	robots	SA	LS	F	F
small instances					
buffalo_free_000_125	125	0.49	0.49	1	3817
galaxy_cluster2_00002_251	251	0.41	0.49	1	9984
algae_00001_278	278	0.38	0.46	1	11771
small_free_017_320	320	0.38	0.46	1	14143
medium_free_003_320	320	0.51	0.52	1	15301
small_018_324	324	0.45	0.53	1	15025
medium_005_407	407	0.57	0.61	1	23608
sun_00001_571	571	0.58	0.60	1	45756
sum		3.77	4.16	8	
medium instances					
clouds_00003_1000	1000	0.39	0.41	1	81966
algae_00004_1113	1113	0.41	0.43	1	99549
medium_017_1202	1202	0.44	0.45	1	124757
buffalo_004_1404	1404	0.40	0.41	1	141613
large_001_1563	1563	0.48	0.49	1	163441
sun_00004_1707	1707	0.41	0.44	1	177165
medium_018_1993	1993	0.41	0.43	1	229762
microbes_00006_2500	2500	0.51	0.51	1	332072
sum		3.45	3.57	8	
large instances					
universe_bgradiation_00006_3000	3000	0.48	0.48	1	452468
algae_00007_4000	4000	0.42	0.42	1	642477
redblue_00009_4500	4500	0.41	0.41	1	776572
galaxy_cluster_00008_5000	5000	0.39	0.39	1	907767
london_night_00009_6000	6000	0.39	0.39	1	1197178
clouds_00009_7229	7229	0.43	0.42	1	1501372
universe_bgradiation_00009_8000	8000	0.44	0.43	1	1737632
large_free_009_9000	9000	0.45	0.44	1	2054727
sum		3.41	3.38	8	

Table 6: A few instances with the scores obtained after running our algorithms for 48 hours.

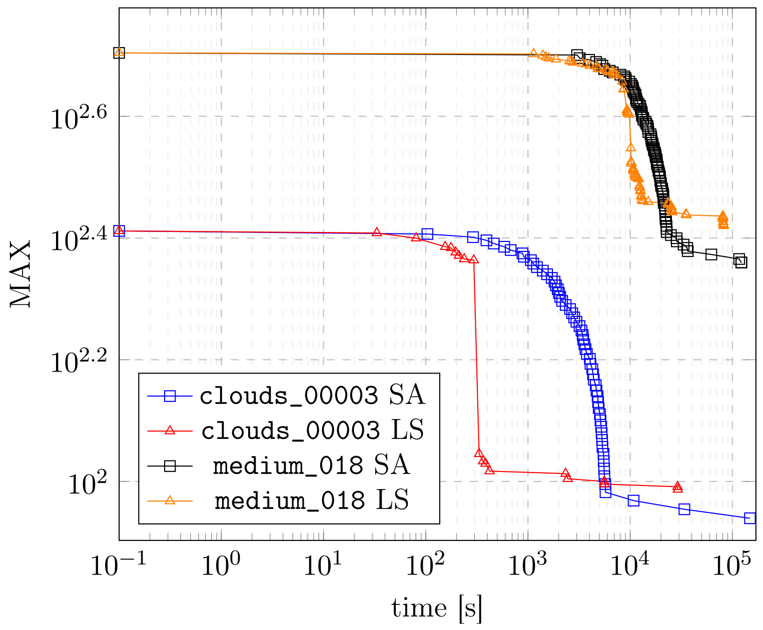


Figure 21: Best makespan until time t for two instances, using LS and SA.

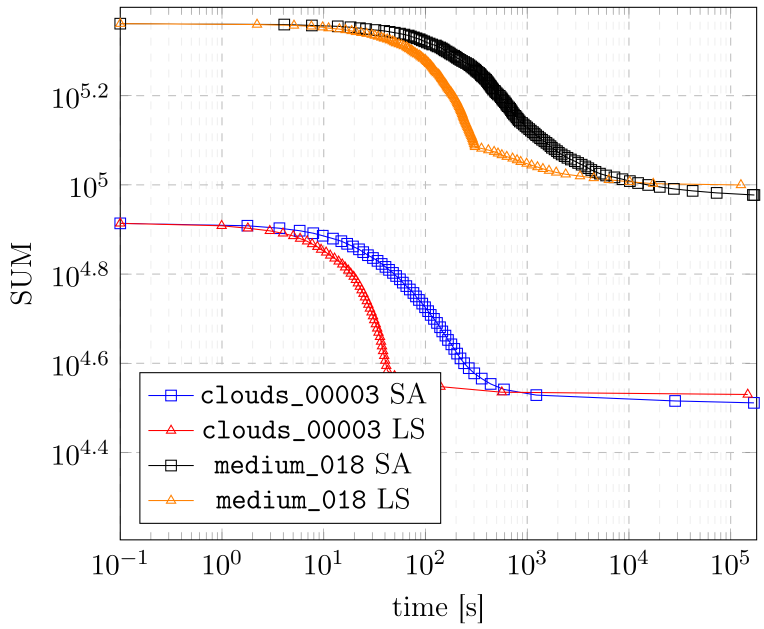


Figure 22: Best SUM until time t for two instances, using LS and SA.

References

[1] H. Alt and L. J. Guibas, "Chapter 3 - Discrete Geometric Shapes: Matching, Interpolation, and Approximation," in *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. North-Holland, 2000, pp. 121–153.

[2] H. Yang and A. Vigneron, "Matching sets of line segments," in *WALCOM: Algorithms and Computation - 13th International Conference, WALCOM 2019, Guwahati, India, February 27 - March 2, 2019, Proceedings*, ser. Lecture Notes in Computer Science, G. K. Das, P. S. Mandal, K. Mukhopadhyaya, and S. Nakano, Eds., vol. 11355. Springer, 2019, pp. 261–273. [Online]. Available: https://doi.org/10.1007/978-3-030-10564-8_21

[3] ——, "Matching sets of line segments," *Theoretical Computer Science*, vol. 866, pp. 82–95, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397521001547

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *J. ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[5] A. Efrat, A. Itai, and M. J. Katz, "Geometry helps in bottleneck matching and related problems," *Algorithmica*, vol. 31, no. 1, pp. 1–28, 2001.

[6] H. Alt, B. Behrends, and J. Blömer, "Approximate matching of polygonal shapes," *Annals of Mathematics and Artificial Intelligence*, vol. 13, no. 3, pp. 251–265, 1995.

[7] L. Chew, M. T. Goodrich, D. P. Huttenlocher, K. Kedem, J. M. Kleinberg, and D. Kravets, "Geometric pattern matching under Euclidean motion," *Computational Geometry*, vol. 7, no. 1, pp. 113–124, 1997.

[8] P. K. Agarwal, M. Sharir, and S. Toledo, "Applications of parametric searching in geometric optimization," *Journal of Algorithms*, vol. 17, no. 3, pp. 292–318, 1994.

[9] A. Efrat, P. Indyk, and S. Venkatasubramanian, "Pattern matching for sets of segments," *Algorithmica*, vol. 40, no. 3, pp. 147–160, 2004.

[10] H. Ahn, O. Cheong, C. Park, C. Shin, and A. Vigneron, "Maximizing the overlap of two planar convex sets under rigid motions," *Comput. Geom.*, vol. 37, no. 1, pp. 3–15, 2007.

[11] S. Cabello, M. de Berg, P. Giannopoulos, C. Knauer, R. van Oostrum, and R. C. Veltkamp, “Maximizing the area of overlap of two unions of disks under rigid motion,” *Int. J. Comput. Geometry Appl.*, vol. 19, no. 6, pp. 533–556, 2009.

[12] S. Har-Peled and S. Roy, “Approximating the maximum overlap of polygons under translation,” *Algorithmica*, vol. 78, no. 1, pp. 147–165, 2017.

[13] A. Vigneron, “Geometric optimization and sums of algebraic functions,” *ACM Trans. Algorithms*, vol. 10, no. 1, pp. 4:1–4:20, 2014.

[14] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl, “Congruence, similarity, and symmetries of geometric objects,” *Discrete & Computational Geometry*, vol. 3, no. 3, pp. 237–256, Sep 1988.

[15] P. J. Heffernan and S. Schirra, “Approximate decision algorithms for point set congruence,” *Computational Geometry*, vol. 4, no. 3, pp. 137 – 156, 1994.

[16] J. Yon, S. Cheng, O. Cheong, and A. Vigneron, “Finding largest common point sets,” *Int. J. Comput. Geometry Appl.*, vol. 27, no. 3, pp. 177–186, 2017.

[17] G. N. Frederickson and D. B. Johnson, “Generalized selection and ranking: Sorted matrices,” *SIAM J. Comput.*, vol. 13, no. 1, pp. 14–30, 1984.

[18] ——, “The complexity of selection and ranking in X+Y and matrices with sorted columns,” *J. Comput. Syst. Sci.*, vol. 24, no. 2, pp. 197–208, 1982.

[19] A. Gajentaan and M. H. Overmars, “On a class of o(n2) problems in computational geometry,” *Comput. Geom.*, vol. 5, pp. 165–185, 1995. [Online]. Available: https://doi.org/10.1016/0925-7721(95)00022-2

[20] G. Barequet and S. Har-Peled, “Polygon containment and translational min-hausdorff-distance between segment sets are 3sum-hard,” *Int. J. Comput. Geometry Appl.*, vol. 11, no. 4, pp. 465–474, 2001. [Online]. Available: https://doi.org/10.1142/S0218195901000596

[21] H. Yang and A. Vigneron, “A Simulated Annealing Approach to Coordinated Motion Planning,” in *37th International Symposium on Computational Geometry (SoCG 2021)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), K. Buchin and E. Colin de Verdière, Eds., vol. 189. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 65:1–65:9. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/13864

[22] ——, “Coordinated path planning through local search and simulated annealing,” *ACM J. Exp. Algorithmics*, apr 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3531224

[23] D. Ratner and M. K. Warmuth, “Finding a shortest solution for the n× n extension of the 15-puzzle is intractable.” in *AAAI*, 1986, pp. 168–172.

[24] R. M. Wilson, "Graph puzzles, homotopy, and the alternating group," *Journal of Combinatorial Theory, Series B*, vol. 16, no. 1, pp. 86–96, 1974.

[25] D. M. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," Master's thesis, M. I. T., Dept. of Electrical Engineering and Computer Science, 1984.

[26] E. D. Demaine, S. P. Fekete, P. Keldenich, H. Meijer, and C. Scheffer, "Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch," *SIAM J. Comput.*, vol. 48, no. 6, pp. 1727–1762, 2019. [Online]. Available: https://doi.org/10.1137/18M1194341

[27] S. P. Fekete, P. Keldenich, D. Krupke, and J. S. B. Mitchell, "Computing coordinated motion plans for robot swarms: The cg:shop challenge 2021," *ACM J. Exp. Algorithmics*, apr 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3532773

[28] L. Crombez, G. D. da Fonseca, Y. Gerard, A. Gonzalez-Lorenzo, P. Lafourcade, and L. Libralesso, "Shadoks approach to low-makespan coordinated motion planning," *ACM J. Exp. Algorithmics*, mar 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3524133

[29] P. Liu, J. Spalding-Jamieson, B. Zhang, and D. W. Zheng, "Coordinated motion planning through randomized k-opt," *ACM J. Exp. Algorithmics*, mar 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3524134

[30] P. van Laarhoven and E. Aarts, *Simulated Annealing: Theory and Applications.* Springer Netherlands, 1987.