Applying Parametric Search to Voting Games and Fréchet Queries

Sampson Wong

School of Computer Science, University of Sydney

A thesis submitted to fulfil requirements for the degree of Master of Philosophy

Abstract

Parametric search, invented by Nimrod Megiddo in 1983, is a complex yet powerful technique for solving general optimisation problems. It has since become a cornerstone technique in computational geometry and has led to efficient algorithms for a wide variety of problems.

In this thesis, we apply parametric search to voting games and Fréchet queries. The connection between voting games and computational geometry is a relatively recent one, and as such, geometric optimisation techniques are not commonly used for solving problems in voting games. We apply parametric search to compute the yolk, which is an important concept in spatial voting games. The connection between the Fréchet distance and parametric search is much more well understood. Our contribution is using repeated inductive applications of parametric search to achieve efficient queries for a variant of a Fréchet distance problem.

Statement of Attribution

Chapter 2 of this thesis is a conference paper published as: Joachim Gudmundsson and Sampson Wong. Computing the yolk in spatial voting games without computing median lines. In *Thirty Third AAAI Conference on Aritificial Intelligence, AAAI 2019*, pages 2012-2019, 2019. I was the corresponding author and the main contributor of the paper.

Chapter 3 of this thesis is an unpublished manuscript to be submitted: Joachim Gudmundsson, André van Renssen, Zeinab Saedi and Sampson Wong. Translation invariant Fréchet distance queries. *Under Submission.* My supervisor was the corresponding author and I was one of the main contributors of the paper.

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Joachim Gudmundsson

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Sampson Wong

Acknowledgements

I thank my supervisor, Joachim Gudmundsson, for his expertise, patience and guidance throughout my Master's candidature. It has been an absolute blast working with you and I look forward to our continued research together.

I would like to extend my thanks to my co-authors and collaborators. Jonathan Chung, André van Renssen and Zeinab Saedi; Vikrant Ashvinkumar, Mees van de Kerkhof, Yuan Sha, Frank Staals, William Umboh and Lionov Wiratma. Our research discussions have been both fun and engaging, and I have learnt so much from you all.

I would also like to thank the rest of the Sydney Algorithms Group for the lunch break chats and the great working atmosphere: Milutin Brankovic, Patrick Eades, Julian Mestre, John Pfeifer and Martin Seybold.

I thank Ralph Holz for agreeing to be my initial auxiliary supervisor.

Contents

List of Figures

Chapter 1

Introduction

Parametric search, invented by Nimrod Megiddo [40], is a complex yet powerful technique for solving general optimisation problems. The technique has since become a cornerstone of geometric optimisation [2]. In computational geometry, parametric search has led to efficient algorithms for the Fréchet distance [5], variants of the Fréchet distance [6, 11, 23], the ham sandwich cut [15], the slope selection problem (or Theil-Sen estimator in statistics) [14], the Euclidean 2-center problem [55], and ray shooting [1], just to name a few.

The technique can be summarised as follows. Let $\lambda \in \mathbb{R}$ be a parameter and let $D(\lambda)$ be a decision problem that evaluates to either true or false for each value of λ. Suppose our optimisation problem is to compute the minimum value of λ so that $D(\lambda)$ evaluates to true. To apply Megiddo's [40] technique, we require the following three properties:

Property 1. The decision problem $D(\lambda)$ is monotone. Formally, if $D(\lambda_0)$ evaluates to true, then $D(\lambda)$ evaluates to true for any $\lambda > \lambda_0$.

Property 2. A decision algorithm $\mathcal{A}(\lambda)$ evaluates the decision problem $D(\lambda)$ for any value of λ. The inputs to $\mathcal{A}(\lambda)$ are the parameter λ along with data objects independent of λ. Let $p_i \in \mathcal{A}(\lambda)$ be a single operation in the algorithm $\mathcal{A}(\lambda)$. If p_i operates only on the data objects independent of λ, then there are no requirements on p_i. If p_i operates on λ as well as the data objects independent of λ_i, then we require that the step p_i be equivalent to making a constant number of comparisons $\{\lambda > c_i\}$ for some constants $\{c_i\}$. The constants c_i can depend only on the data objects and not on the parameter λ. The constants c_i are called the critical values associated with the step $p_i \in \mathcal{A}(\lambda)$.

Property 3. The decision algorithm $A(\lambda)$ has a serial running time of T_s. Given P processors, the decision algorithm $A(\lambda)$ has a parallel running time of T_p per processor.

If the above three properties hold, then the technique states that there is an $O(PT_p + T_pT_s \log P)$ time algorithm to compute the minimum value of λ so that $D(\lambda)$ evaluates to true. This running time is usually only a polylogarithmic factor slower than the running time of the decision algorithm. The stated running time is a serial running time, but interestingly, it depends on the parameters P and T_p which are the parallel processors and parallel running times of the decision algorithm respectively. The proof of correctness for this optimisation algorithm can be found in [2]. There are two noteworthy extensions to the parametric search technique.

The first extension is Cole's extension of parametric search [13]. This extension states that the running time of the optimisation algorithm of Megiddo's can be improved by a logarithmic factor to $O(PT_p + T_sT_p)$, if following condition is met: that all p_i that generate the critical values c_i are part of sorting operations, and that all such sorting operations can be replaced by the AKS parallel sorting scheme [3]. We do not use Cole's extension in this thesis but we suspect that it may be able to be applied to some of our results.

The second extension is that parametric search can be used to solve multidimensional problems. Multidimensional parametric search is the main technique we use in Chapter 2, and we give an overview of the technique in that chapter.

In this thesis, we apply parametric search to voting games and to Fréchet queries. The connection between voting games and computational geometry is a relatively recent one, and as such, geometric optimisation techniques are not commonly used for solving problems in voting games. We apply parametric search to compute the yolk, which is an important concept in spatial voting games. The connection between the Fréchet distance and parametric search is much more well understood. Our contribution is using repeated inductive applications of parametric search to achieve efficient queries for a variant of a Fréchet distance problem.

For voting games we focus on a geometric setting, referred to as the spatial model of voting. In this model, the yolk is an important concept and has connections to the pure Nash equilibrium and the uncovered set. In Chapter 2, we present near-linear time algorithms for computing the yolk in the plane. To the best of our knowledge our algorithm is the first that does not precompute median lines, and hence is able to break the best known upper bound of $O(n^{4/3})$ given by the number of limiting median lines.

In Chapter 3, we consider a well studied variant of the Fréchet distance known as the Translation Invariant Fréchet distance. This variant is motivated by some applications where it is desirable to match the curves under translation before computing the Fréchet distance between them. Algorithms to compute the Translation Invariant Fréchet distance are known [6, 11, 33, 67], however the query version is much less well understood. We study Translation Invariant Fréchet distance queries in a restricted setting of horizontal query segments. More specifically, we preprocess a trajectory in $O(n^2 \log^2 n)$ time and space, such that for any subtrajectory and any horizontal query segment we can compute their Translation Invariant Fréchet distance in $O(\operatorname{polylog} n)$ time.

Chapter 2

Computing the Yolk in Spatial Voting Games

Voting theory is concerned with preference aggregation and group decision making. A classic framework for aggregating voter's preferences is the Downsian [20], or spatial model of voting.

In this model, voters are positioned on a 'left-right' continuum along multiple ideological dimensions, such as economic, social or religious. These dimensions together form the policy space. Each voter is required to choose a single candidate from a set of candidates, and a common voter preference function is a metric/distance function within the policy space. An intuitive reason behind using metric preferences is that voters tend to prefer candidates ideologically similar to themselves.

The spatial model of voting with metric preferences have been studied extensively, both theoretically [22, 37, 38, 43, 62] and empirically [44, 46, 47, 48, 52, 53, 54]. Recently, lower bounds were provided on the distortion of voting rules in the spatial model, and interestingly, metrics other than the Euclidean metric were considered [7, 30, 56].

We focus our attention on two-candidate spatial voting games, where the winner is the candidate preferred by a simple majority of voters. In a one dimension policy space, Black's Median Voter Theorem [8] states that a voting equilibrium (alt. Condorcet winner, plurality point, pure Nash equilibrium) is guaranteed to exist and coincides with the median voter.

Naturally, social choice theorists searched for the equilibrium in the two dimensional policy space, but these attempts were shown to be fruitless [45]. The initial reaction was one of cynicism [37], but in response a multitude of generalisations were developed, with the yolk being one such concept [38, 43]. The yolk in the Euclidean $\mathcal{L}_2$ metric is defined as the minimum radius disk that intersects all median lines of the voters.

The yolk is an important concept in spatial voting games due to its simplicity and its relationship to other concepts. The yolk radius provides approximate

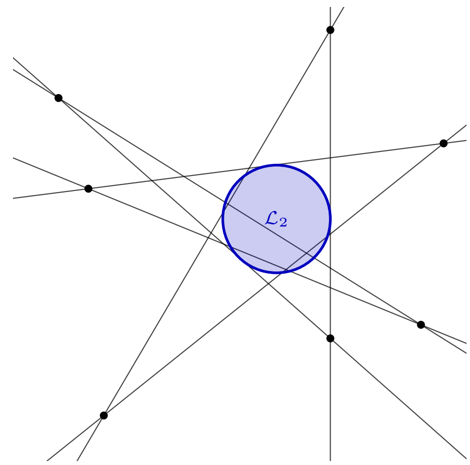


Figure 2.1: The $\mathcal{L}_2$ yolk intersects all median lines of voters.

bounds on the uncovered set [25, 42, 43], limits on agenda control [27], Shapley-Owen power scores [24], the Finagle point [68] and the ε-core [65]. As such, studies on the size of the yolk [26, 35, 63] translate to these other concepts as well.

From the perspective of computational social choice, this raises the following problem: Are there efficient algorithms for computing the yolk? Fast algorithms would, for instance, facilitate empirical studies on large data sets. Tovey [61] provides the first polynomial time algorithm, which in two dimensions, computes the yolk in $O(n^4)$ time. De Berg et al. [17] provides an improved $O(n^{4/3} \log^{1+\varepsilon} n)$ time algorithm for the same.

The shortcoming of existing algorithms is that they require the computation of all limiting median lines, which are median lines that pass through at least two voters [58]. However, there are $\Omega(ne^{\sqrt{\log n}})$ [60] limiting median lines in the worst case. Moreover, the best known upper bound of $O(n^{4/3})$ seems difficult to improve on [19]. It is an open problem as to whether there is a faster algorithm that computes the yolk without precomputing all limiting median lines.

Problem Statement

Given a set V of n points in the plane, a median line of V is any line that divides the plane into two closed halfplanes, each with at most $n/2$ points. The yolk is a minimum radius disk in the $\mathcal{L}_p$ metric that intersects all median lines of V.

We compute yolks in the $\mathcal{L}_1$ (Taxicab), the $\mathcal{L}_2$ (Euclidean), and the $\mathcal{L}_\infty$ (Uniform) metrics. As shown in Figure 2.2, the yolk in $\mathcal{L}_1$ is the smallest 45°-rotated square and in $\mathcal{L}_\infty$ the smallest axis-parallel square, that intersects all median lines of V.

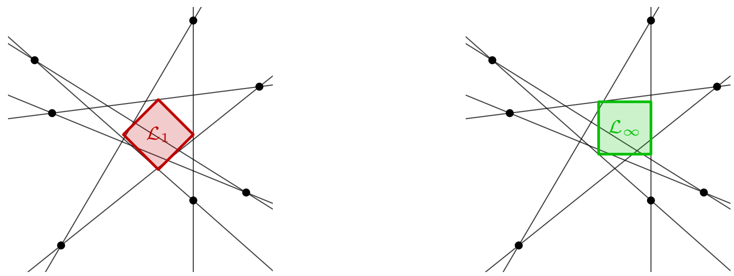


Figure 2.2: Example of yolks in the $\mathcal{L}_1$ and $\mathcal{L}_\infty$ metrics.

Our Contribution and Results

Our contributions are, first, an algorithm that computes the yolk in the $\mathcal{L}_1$ and $\mathcal{L}_\infty$ metrics in $O(n \log^7 n)$ time, and second, an algorithm that computes a $(1+\varepsilon)$-approximation of the yolk in the $\mathcal{L}_2$ metric in $O(n \log^7 n \cdot \log^4 \frac{1}{\varepsilon})$ time.

We achieve the improved upper bounds by carefully applying Megiddo's [40] parametric search technique, which is a powerful yet complex technique and that could be useful for other spatial voting problems.

The parametric search technique is a framework for converting decision algorithms into optimisation algorithms. For the yolk problem, a decision algorithm would decide whether a given disk intersects all median lines. If this decision algorithm satisfies the three properties as specified by the framework, then Megiddo's result states that there is an efficient algorithm to compute the yolk.

For the purposes of designing a decision algorithm with the desired properties, we instead consider the more general problem of finding the smallest regular, k-sided polygon that intersects all median lines of V. The regular k-sided polygon $P_k(r, x, y)$ is shown in Figure 2.3 and is defined as:

Definition 1. Given an integer $k \geq 3$, construct the regular k-sided polygon $P_k(r, x, y)$ by:

- Constructing a circle with radius r and centered at (x, y).
- Placing a vertex at the top-most point on the circle, i.e. at $(x, y + r)$.
- Placing the remaining $k-1$ vertices around the circle so that the k vertices are evenly spaced.

In Section 2.1, we present the decision algorithm, which given a regular, k-sided polygon $P_k(r, x, y)$, decides whether the polygon intersects all median lines of V. Next, in Section 2.2, we apply Megiddo's technique to the decision algorithm and prove the convexity and parallelisability properties. This leaves one final property, the existence of critical hyperplanes, left to check. We prove this final property in Sections 4-6, thus completing the parametric search. Finally, in Section 7, we show that our general problem for the regular, k-sided polygon $P_k(r, x, y)$ implies the claimed running times by setting $k = 4$ for $\mathcal{L}_1$ and $\mathcal{L}_\infty$, and $k = \frac{1}{\varepsilon}$ for $\mathcal{L}_2$.

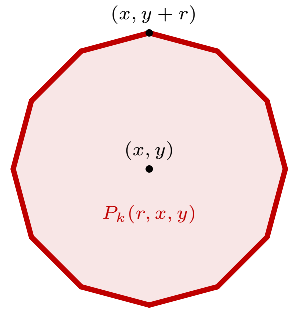


Figure 2.3: The regular, k-sided polygon $P_k(r, x, y)$.

2.1 Decision Algorithm

The aim of this section is to design an algorithm that solves the following decision problem:

Definition 2. Given an integer $k \geq 3$ and a set V of n points in the plane, the decision problem $D_{k,V}(r, x, y)$ is to decide whether the polygon $P_k(r, x, y)$ intersects all median lines of V.

We show that there is a comparison-based decision algorithm that solves $D_{k,V}(r, x, y)$ in $O(n \log n \cdot \log k)$ time, provided the following two comparison-based subroutines.

Subroutine 1. A comparison-based subroutine that, given a point p and a regular k-sided polygon $P_k(r, x, y)$, in $O(\log k)$ time decides if p is outside $P_k(r, x, y)$.

Subroutine 2. A comparison-based subroutine that, given points p, q outside a regular k-sided polygon $P_k(r, x, y)$, in $O(\log k)$ time sorts in a clockwise order the four tangent lines drawn through $\{p, q\}$ and tangent to $P_k(r, x, y)$.

Although the running time of these two subroutines are not too difficult to prove, we shall see in Section 3 that these subroutines must satisfy a stronger requirement for the parametric search technique to apply. We will formally define the stronger requirement in the next section. To avoid repetition, we simultaneously address the subroutine and the stronger requirement in Sections 5 and 6. But for now, we assume the subroutines exist and present the decision algorithm:

Theorem 1. Given an integer $k \geq 3$ and a set V of n points in the plane, there is a comparison-based algorithm that solve the decision problem $D_{k,V}(r, x, y)$ in $O(n \log n \cdot \log k)$ time, provided that Subroutine 1 and Subroutine 2 exist.

Proof. The proof comes in three parts. First, we transform the decision problem $D_{k,V}(r, x, y)$ into an equivalent form that does not have median lines in its statement. Then, we present a sweep line algorithm for the transformed version. Finally, we perform an analysis of the running time.

Consider for now a single median line m_g that has gradient g. Construct two parallel lines $t_U(g)$ and $t_D(g)$ that also have gradient g, but are tangent to $P_k(r, x, y)$ from above and below respectively. If the median line m_g intersects $P_k(r, x, y)$, as shown in Figure 2.4, then m_g must be in between $t_U(g)$ and $t_D(g)$.

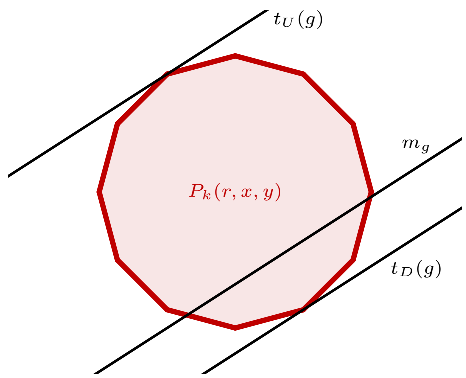


Figure 2.4: The relative positions of m_g, $t_U(g)$ and $t_D(g)$ if m_g intersects the k-sided regular polygon $P_k(r, x, y)$.

We will decide whether all median lines of gradient g are between $t_U(g)$ and $t_D(g)$, as this would immediately decide whether all median lines of gradient g intersects $P_k(r, x, y)$. We will solve this restricted decision problem by counting the number of points in V above $t_U(g)$ and the number of points in V below $t_D(g)$.

Let $t_U^+(g)$ be the number of points in V that are above $t_U(g)$, and similarly $t_D^-(g)$ for the points in V below $t_D(g)$. Suppose that $t_U^+(g) < n/2$ and $t_D^-(g) < n/2$. Then there cannot be a median line of gradient g above $t_U(g)$ or below $t_D(g)$, since one side of the median line, in particular the side that contains the polygon, will have more than $n/2$ points. Hence, if $t_U^+(g) < n/2$ and $t_D^-(g) < n/2$, then all median lines of gradient g must be between $t_U(g)$ and $t_D(g)$.

Conversely, suppose that all median lines of gradient g are between $t_U(g)$ and $t_D(g)$. Then if $t_U^+(g) \geq n/2$, we can move $t_U(g)$ continuously upwards until it becomes a median line, which is a contradiction. So in this case, we know $t_U^+(g) < n/2$ and $t_D^-(g) < n/2$.

In summary, we have transformed the decision problem into one that does not have median lines in its statement: All median lines intersect $P_k(r, x, y)$ if for all gradients g, the pair of inequalities $t_U^+(g) < n/2$ and $t_D^-(g) < n/2$ hold.

We present a sweep line algorithm that computes whether the pair of inequalities hold for all gradients g. Let t be an arbitrary line tangent to the polygon $P_k(r, x, y)$, and define t^+ to be the open halfplane that has t as its boundary and does not include the polygon $P_k(r, x, y)$. Then all median lines intersect $P_k(r,x,y)$ if and only if for all positions of t, the open halfplane t^+ contains less than $n/2$ points.

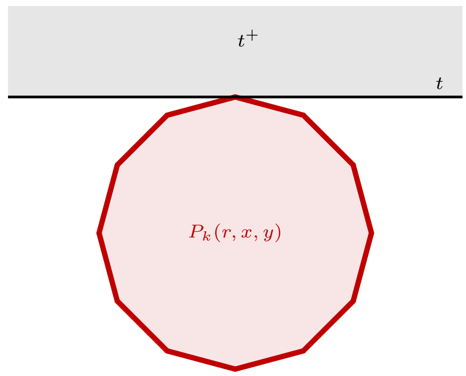


Figure 2.5: The rotating sweepline t and the open halfplane t^+.

The tangent line t is a clockwise rotating sweep line and the invariant maintained by the sweep line algorithm is the number of points of V inside the region t^+. Take any tangent line t_0 to be the starting line, and calculate the number of points in t_0^+. From here, define an event to be when the line t passes through a point. There are two events for each point outside $P_k(r,x,y)$; there is one event for when the point enters the region t^+, and one for when it exits. There are no events for points of V that lie inside $P_k(r,x,y)$.

The sweepline algorithm first computes the set of events, then sorts the set of events, and finally processes each event one by one.

First use Subroutine 1 to decide which points of V are outside $P_k(r,x,y)$. Each point outside of V gives two events, as noted in the paragraph above. This takes $O(n \log k)$ time in total. Next, we sort the set of events. To decide which of the two events occur first in the clockwise order, we only need to make a single call to Subroutine 2, which takes $O(\log k)$ time. To completely sort all $O(n)$ events, we require an efficient comparison-based sorting algorithm, for example Merge sort, which will make $O(n \log n)$ calls to Subroutine 2. This takes $O(n \log n \log k)$ time in total. Finally, we process the events one by one to maintain our invariant, which we recall is the number of points of V inside the region t^+. This value increases by one at "entry" events and decreases by one at "exit" events. This takes $O(n)$ time. After processing all events we return whether our invariant remained less than $n/2$ at all events. The total running time is dominated by sorting the set of events, which takes $O(n \log n \cdot \log k)$ time. □

2.2 Parametric Search

Parametric search is a powerful yet complex technique for solving optimisation problems. The two steps involved in this technique are, first, to design a decision algorithm, and second, to convert the decision algorithm into an optimisation algorithm.

For example, our parameter space is $(r, x, y) \in \mathbb{R}^3$, our decision algorithm is stated in Theorem 1, and our optimisation objective is to minimise $r \in \mathbb{R}^+$.

2.2.1 Preliminaries

Megiddo's (1983) states the requirements for converting the decision algorithm into an optimisation algorithm. First, let us introduce some notation. Let $\mathbb{R}^d$ be a parameter space, let $\lambda \in \mathbb{R}^d$ be a parameter and let $D(\lambda)$ be a decision problem that either evaluates to true or false. Then the first requirement is for the decision problem $D(\lambda)$.

Property 1. The set of parameters $\{\lambda \in \mathbb{R}^d : D(\lambda)\}$ that satisfies the decision problem is convex.

Convexity guarantees that the optimisation algorithm finds the global optimum.

The second property of the technique relates to the decision algorithm. Let $\mathcal{A}(\lambda)$ be a comparison-based decision algorithm that computes $D(\lambda)$. Let $C(\lambda)$ be any comparison in the comparison-based decision algorithm $\mathcal{A}(\lambda)$. The comparison $C(\lambda)$ is said to have an associated critical hyperplane in $\mathbb{R}^d$ if the result of the comparison is linearly separable with respect to $\lambda \in \mathbb{R}^d$. Formally, suppose that the comparison $C(\lambda)$ evaluates to either $>$, $=$ or $<$. Then we say that the $(d-1)$-dimensional hyperplane $H \subset \mathbb{R}^d$ is the associated critical hyperplane of $C(\lambda)$ if C evaluates to $>$, $=$ or $<$ if and only if λ is above, on, or below H respectively. The comparisons of the decision algorithm must satisfy the following property.

Property 2. Every comparison $C(\lambda)$ in the comparison-based decision algorithm $\mathcal{A}(\lambda)$ either (i) does not depend on λ, or (ii) has an associated critical hyperplane in $\mathbb{R}^d$.

This requirement allows us to compute a large set of critical hyperplanes that determines the result of $\mathcal{A}(\lambda)$. Moreover, the optimum must lie on one of these critical hyperplanes, since the result of $\mathcal{A}(\lambda)$ locally changes sign at the optimum. The new search space now has dimension $d-1$ instead of dimension d, and we can recursively apply this procedure to reduce the dimension further. For details see [2].

The final property speeds up the parametric search.

Property 3. The decision algorithm has an efficient parallel algorithm.

If the decision algorithm $\mathcal{A}(\lambda)$ runs in T_s time and runs on P processors in T_p parallel steps, then the parametric search over $\lambda \in \mathbb{R}^d$ runs in $O(T_pP + T_s(T_p \log P)^d)$ time [2].

2.2.2 Applying the technique

To apply the parametric search technique, we show that our decision problem $D_{k,V}(r, x, y)$ satisfies Properties 1-3.

Lemma 1. Given an integer $k \geq 3$ and a set V of n points in the plane, the set of parameters $\{(r, x, y) : D_{k,V}(r, x, y)\}$ that satisfies the decision problem is convex.

Proof. Suppose we are given a convex combination $\lambda_3 = \alpha\lambda_1 + (1-\alpha)\lambda_2$ of the two parameters $\lambda_1, \lambda_2 \in \mathbb{R}^3$. Then the polygon $P_k(\lambda_3)$ is a convex combination of the polygons $P_k(\lambda_1)$ and $P_k(\lambda_2)$. It is easy to check that if a line m intersects both $P_k(\lambda_1)$ and $P_k(\lambda_2)$, then the line m must also intersect the convex combination $P_k(\lambda_3)$.

Now assume that both $D_{k,V}(\lambda_1)$ and $D_{k,V}(\lambda_2)$ are true. Then for any median line m both $P_k(\lambda_1)$ and $P_k(\lambda_2)$ intersect m. By the observation above, the convex combination $P_k(\lambda_3)$ must also intersects m. Repeating this fact for all median lines implies that $P_k(\lambda_3)$ intersects all median lines of V. So $D_{k,V}(\lambda_3)$ is true whenever $D_{k,V}(\lambda_1)$ and $D_{k,V}(\lambda_2)$ are true. Therefore, the set of parameters $\{(r, x, y) \subseteq \mathbb{R}^3 : D_{k,V}(r, x, y)\}$ is convex. $\square$

Lemma 2. Every comparison in the decision algorithm in Theorem 1 either (i) does not depend on (r, x, y), or (ii) has an associated critical hyperplane in $\mathbb{R}^3$.

Proof. Theorem 1 consists of three steps, computing the points outside the polygon, computing the event order, and processing the events. For the first two steps, the comparisons do depend on (r, x, y) and have associated critical hyperplanes. We defer the proof of this claim to Sections 2.4 and 2.5 respectively. For the third step, the comparisons do not depend on (r, x, y) but rather the event order, so there is no requirement that comparisons have critical hyperplanes. $\square$

Lemma 3. Given n processors, the decision algorithm in Theorem 1 has a parallel running time of $O(\log n \cdot \log k)$ per processor.

Proof. Given n processors, we show how to parallelise the key steps of the decision algorithm in Theorem 1. The key steps are computing the events, sorting the events, and processing the events. For computing the events, we need to decide which points are outside the polygon with Subroutine 1. If we assign one processor to each point of V then the parallel running time is $O(\log k)$. For sorting the events, instead use Preparata's sorting scheme [49], which states that a set of n objects can be sorted with n processors in $O(\log n)$ comparisons per processor. Since each processor makes $O(\log n)$ comparisons, and by Subroutine 2 each comparison takes $O(\log k)$ time, the parallel running time per processor is $O(\log n \log k)$. Finally, processing the events generates no critical hyperplanes so this step does not require parallelisation. $\square$

Now we combine Properties 1-3 with Megiddo's result to obtain an optimisation algorithm for the smallest, regular, k-sided polygon $P_k(r, x, y)$ that intersects all median lines.

Theorem 2. Given a set V of n points in the plane, there is an $O(n \log^7 n \cdot \log^4 k)$ time algorithm to compute the minimum r such that $D_{k,V}(r,x,y)$ is true for some regular, k-sided polygon $P_k(r,x,y)$.

Proof. Megiddo's multidimensional parametric search implies that there is an efficient optimisation algorithm. It only remains to compute the running time of the technique.

The parallel algorithm runs on $P = O(n)$ processors in $T_p = O(\log n \cdot \log k)$ parallel steps, whereas the decision algorithm runs in $T_s = O(n \log n \cdot \log k)$ time. The dimension d of the parameter space is three. The running time of multidimensional parametric search is $O(T_p P + T_s (T_p \log P)^d)$ [2]. Substituting our values into the above formula yields the required bound. □

2.3 Computing Critical Hyperplanes

The only requirement left to check is Property 2 for the comparisons in the comparison-based subroutines, that is, Subroutine 1 and Subroutine 2. Before launching into the analysis of the two subroutines, we first prove a tool. We will use the tool repeatedly in the next two sections to simplify checking Property 2.

Lemma 4. Let gradient $g \in \mathbb{R}$, point $p \in \mathbb{R}^2$ and vector $v \in \mathbb{R}^2$ be given, and let $(r,x,y) \in \mathbb{R}^3$ be a variable parameter. Let $L_{g,v}(r,x,y)$ be the line of gradient g through the point $(x,y) + r \cdot v$. Then there exists a hyperplane $H_{p,g,v}$ of $\mathbb{R}^3$ such that p is above, on, or below $L_{g,v}(r,x,y)$ if and only if the point (r,x,y) is above, on, or below $H_{p,g,v}$.

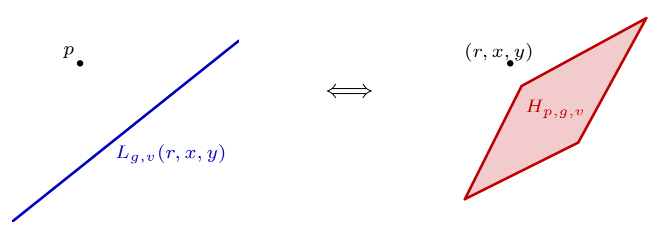


Figure 2.6: Point p is above $L_{g,v}(r,x,y)$ if and only if parameter (r,x,y) is above $H_{p,g,v}$.

Proof. Let point $p = (p_x, p_y)$ and vector $v = (v_x, v_y)$. Now, (p_x, p_y) is above the line through (q_x, q_y) of gradient g if and only if $(p_y - q_y) - g \cdot (p_x - q_x) > 0$. Substituting the point $(q_x, q_y) = (x,y) + r \cdot (v_x, v_y)$, we get the inequality

$$(p_y - y - rv_y) - g \cdot (p_x - x - rv_x) > 0.$$

This inequality can be rearranged into the form $ax + by + cr + d > 0$, where

$$a = g, \; b = -1, \; c = (gv_x - v_y), \; d = p_y - gp_x.$$

In this form, we can see that the inequality is satisfied if and only if (r, x, y) lies above the hyperplane $H_{p,g,v} := (ax + by + cr + d = 0)$, where a, b, c, d are given above. Hence, checking if p is above line $L_{g,v}(r, x, y)$ is equivalent to checking if (r, x, y) is above $H_{p,g,v}$, as required. □

Many of the comparisons in Sections 2.4 and 2.5 will be of the form as stated in the lemma above. For these comparisons, we say that $H_{p,g,v}$ is its associated critical hyperplane. Now we are ready to address the subroutines.

2.4 Subroutine 1

Subroutine 1 decides whether a given point p is outside the k-sided, regular polygon $P_k(r, x, y)$. We present an $O(\log k)$ time comparison-based algorithm and show that Property 2 holds.

Lemma 5. Subroutine 1 has an $O(\log k)$ time comparison-based algorithm, and comparisons in the algorithm that depend on the parameter (r, x, y) each have an associated critical hyperplane.

Proof. We partition the polygon $P_k(r, x, y)$ into k triangles, and decide which partition the point p is in, if it indeed is in any of these partitions. For $1 \leq i \leq k$, the i^{th} partition of $P_k(r, x, y)$ is the triangle joining the i^{th} vertex, the $(i + 1)^{th}$ vertex and the center of $P_k(r, x, y)$. Figure 2.7 shows the i^{th} partition of $P_k(r, x, y)$.

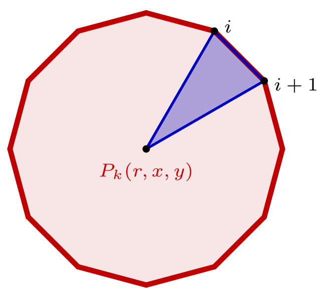


Figure 2.7: The i^{th} partition of $P_k(r, x, y)$.

Assume for now that the point p is indeed in the polygon $P_k(r, x, y)$ and hence in one of the k partitions. We decide whether p is in the i^{th} partition for some $i \leq j$, or for some $i > j$, and perform a binary search for the index i. This can be done by deciding if the point p is above, on, or below the line joining the center of $P_k(r, x, y)$ and its j^{th} vertex. The comparison depends on (r, x, y), so we must compute its associated critical hyperplane using Lemma 4. Let $P_k(1, 0, 0)$ be the k-sided polygon of radius 1 and centered at the origin. Then set g to be the gradient of the line joining the center to the i^{th} vertex of $P_k(1, 0, 0)$, and vector $v = 0$ in Lemma 4 to obtain the associated critical hyperplane.

We have searched for the partition that p is in if it is indeed in $P_k(r, x, y)$. Hence, it only remains to decide whether p is indeed in that partition. This requires a constant number of comparisons, each of which depend on (r, x, y). We have already computed associated critical hyperplanes for two of the sides. The last side joins two adjacent vertices of the polygon $P_k(r, x, y)$. Set g to be the gradient of the i^{th} side of polygon $P_k(1, 0, 0)$, and the vector v to be the i^{th} vertex of $P_k(1, 0, 0)$, to obtain the final associated critical hyperplane.

The running time is dominated by the binary search for the i^{th} partition, which takes $O(\log k)$ time. □

2.5 Subroutine 2

Subroutine 2 computes the relative clockwise order of four tangent lines drawn from two points to polygon $P_k(r, x, y)$.

Lemma 6. Subroutine 2 has an $O(\log k)$-time comparison-based algorithm, and comparisons in the algorithm that depend on the parameter (r, x, y) each have an associated critical hyperplane.

Proof. Draw two lines t_i, t_j tangent to $P_k(r, x, y)$ and parallel to pq, and let the points of tangency be vertex i and vertex j. If there are multiple points of tangency then choose any such point. Then without loss of generality, set ij to be horizontal, and assume further that p has a larger y coordinate than q. Then the t_i, t_j and ij partition the plane into the four regions, as shown in Figure 2.8. Region L is left of both tangents, R is right of both tangents, U is between the tangents and above ij, and D is between the tangents and below ij.

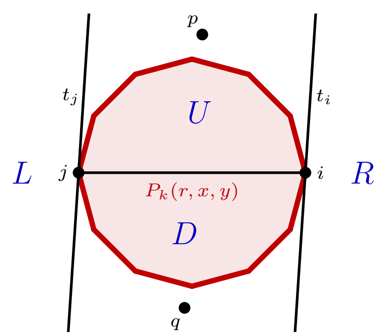


Figure 2.8: The lines t_i, t_j, ij partition the plane into regions L, R, U, D.

Then the relative clockwise order of the four lines drawn from p and q are determined by which of the four regions L, R, U or D the points p and q are located. See Figure 2.9.

Five cases follows. Let p_e and p_x points of tangency from p such that the points p_e, p, p_x are in clockwise order. If p, q are in the same region, then the containing region L, R, U, and D correspond to the relative clockwise

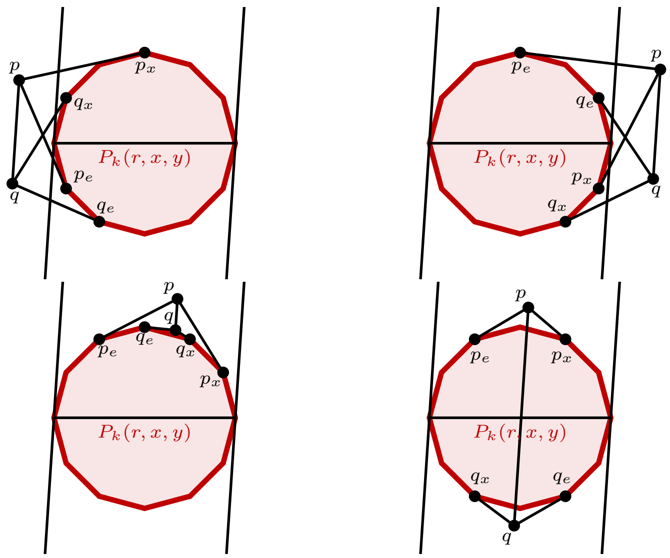


Figure 2.9: The relative orders shown for when (i) $p, q \in L$, (ii) $p, q \in R$, ((iii) $p, q \in U$ and (iv) $p \in U, q \in D$.

orders $q_e p_e q_x p_x$, $p_e q_e p_x q_x$, $p_e q_e q_x p_x$, and $q_e p_e p_x q_x$ respectively. If p, q are in different regions, then they must be in U and D respectively, and the relative order is $p_e p_x q_e q_x$. The proof for case analysis for the five cases is omitted, but the diagrams in Figure 2.9 may be useful for the reader.

The running time of the algorithm is as follows. Given the gradient of pq, there is an $O(\log k)$ time algorithm to binary search the gradients of the sides of $P_k(r, x, y)$ to compute the vertices i and j. Then the remainder of the algorithm takes constant time: rotating the diagram so that ij is horizontal, deciding whether p or q has a larger y coordinate, and computing the region L, R, U, D that points p, q are in.

The proof of existence of critical hyperplanes is as follows. Since the gradients of pq and sides of P_k do not depend on (r, x, y), computing i and j generates no critical hyperplanes. Similarly, rotating the diagram so that ij is horizontal and then deciding which of p or q have larger y coordinates also generates no critical hyperplanes. It only remains to decide which of the four regions L, R, U, D the point p, and respectively q, is in. Set g to the gradient of pq and vector v to be the i^{th} vertex of $P_k(1, 0, 0)$ in Lemma 4 to decide if p is to the left of the tangent through i. Do so similarly for j to decide if p is to the right of the tangent through j. Finally, set g to the gradient of ij and vector v to be either the i^{th} or j^{th} vertex of $P_k(1, 0, 0)$ to decide if p is above the chord ij. $\square$

Checking that Property 2 holds for the comparison-based subroutines, Subroutine 1 and Subroutine 2, completes the proof to Theorem 2. In the final section we will prove that Theorem 2 implies that we have an efficient algorithm for computing the yolk in the $\mathcal{L}_1$ and $\mathcal{L}_\infty$ metrics, and an efficient approximation algorithm for the $\mathcal{L}_2$ metric.

2.6 Computing the Yolk in $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_\infty$

It remains to show that our general problem for $P_k(r, x, y)$ implies the results as claimed in the introduction.

Theorem 3. Given a set V of n points in the plane, there is an $O(n \log^7 n)$ time algorithm to compute the yolk of V in the $\mathcal{L}_1$ and $\mathcal{L}_\infty$ metrics.

Proof. Setting $k = 4$ in Theorem 2 gives an algorithm to compute the smallest $P_4(r, x, y)$ that intersects all median lines of V in $O(n \log^7 n)$ time. This rotated square coincides with yolk in the $\mathcal{L}_1$ metric, refer to Figure 2.2 and Definition 1.

Computing the yolk in the $\mathcal{L}_\infty$ metric requires one extra step. Rotate the points of V by 45° clockwise, compute the smallest $P_4(r, x, y)$, and then rotate the square $P_4(r, x, y)$ back 45° anticlockwise to obtain the yolk in the $\mathcal{L}_\infty$ metric. □

Theorem 4. Given a set V of n points in the plane and an $\varepsilon > 0$, there is an $O(n \log^7 n \cdot \log^4 \frac{1}{\varepsilon})$ time algorithm to compute a $(1+\varepsilon)$-approximation of the yolk in the $\mathcal{L}_2$ metric.

Proof. Set $k = \lceil \pi \cdot (1 + \frac{1}{\varepsilon}) \rceil$. Theorem 2 gives an algorithm to compute the smallest $P_k(r, x, y)$ that intersects all median lines of V in the desired running time. It suffices to show that for this parameter set (r, x, y), the disk centered at (x, y) with radius r is a $(1 + \varepsilon)$-approximation for the yolk in the $\mathcal{L}_2$ metric.

First, note that $P_k(r, x, y)$ intersects all median lines, and $B(r, x, y)$ encloses $P_k(r, x, y)$, so the disk must also intersect all median lines of V. Hence, it remains to show that the radius r of $B(r, x, y)$ satisfies $r \leq (1+\varepsilon) \cdot r_2$, where r_2 is the radius of the true yolk in the $\mathcal{L}_2$ metric.

Let the yolk in the $\mathcal{L}_2$ metric be the disk $B(r_2, x_2, y_2)$. Consider the regular, k-sided polygon $P_k(r_2 \cdot \sec \frac{\pi}{k}, x_2, y_2)$, so that by construction, all sides of this polygon are tangent to $B(r_2, x_2, y_2)$.

Now since $B(r_2, x_2, y_2)$ is the $\mathcal{L}_2$ yolk, it intersects all median lines and so does its enclosing polygon $P_k(r_2 \cdot \sec \frac{\pi}{k}, x_2, y_2)$. By the minimality of $P_k(r, x, y)$, we get $r \leq \sec \frac{\pi}{k} \cdot r_2$. But for $\theta \in [0, \frac{\pi}{3}]$, we have $\sec \theta \leq \frac{1}{1-\theta}$. So,

$$\sec \frac{\pi}{k} \leq \frac{1}{1 - \frac{\pi}{k}} \leq 1 + \varepsilon,$$

which implies that $r \leq (1 + \varepsilon) \cdot r_2$, as required. □

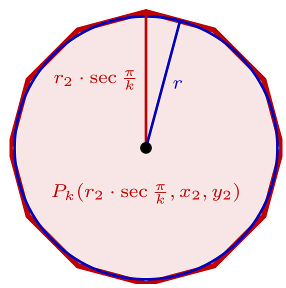


Figure 2.10: The polygon $P_k(r_2 \cdot \sec \frac{\pi}{k}, x_2, y_2)$ is externally tangent to the disk $B(r_2, x_2, y_2)$.

2.7 Concluding Remarks

Cole's [13] extension to parametric search states that the running time of the parametric search may be reduced if certain comparisons are delayed. This is a direction for further research that could potentially improve the running time of our algorithms.

An open problem is whether one can compute the yolk in higher dimensions without precomputing all median hyperplanes. Avoiding the computation of median hyperplanes yields even greater benefits as less is known about bounds on the number of median hyperplanes in higher dimensions.

Similarly, our approximation algorithm for the $\mathcal{L}_2$ yolk in the plane is optimal up to polylogarithmic factors, however, it is an open problem as to whether there is a near-linear time exact algorithm. Our attempts to apply Megiddo's parametric search technique to the $\mathcal{L}_2$ yolk have been unsuccessful so far.

Finally, there are other solution concepts in computational spatial voting that currently lack efficient algorithms. The shortcomings of existing algorithms are: for the Shapley-Owen power score there is only an approximate algorithm [29], for the Finagle point only regular polygons have been considered [68] and for the ε-core only a membership algorithm exists [64]. Since these problems have a close connection to either median lines or minimal radius, we suspect that Megiddo's parametric search technique may be useful.

Chapter 3

Translation Invariant Fréchet Distance Queries

The Fréchet distance is a popular measure of similarity between curves as it takes into account the location and ordering of the points along the curves, and it was introduced by Maurice Fréchet in 1906 [28]. Measuring the similarity between curves is an important problem in many areas of research, including computational geometry [5, 9, 21], computational biology [33, 69], data mining [34, 51, 66], image processing [4, 57] and geographical information science [36, 41, 50, 59].

The Fréchet distance is most commonly described as the dog-leash distance; consider a man standing at the starting point of one trajectory and the dog at the starting point of another trajectory. A leash is required to connect the dog and its owner. Both the man and his dog are free to vary their speed, but they are not allowed to go backward along their trajectory. The cost of a walk is the maximum leash length required to connect the dog and its owner from the beginning to the end of their trajectories. The Fréchet distance is the minimum length of the leash that is needed over all possible walks. More formally, for two curves A and B each having complexity n, the Fréchet distance between A and B is defined as:

$$\delta_F(A, B) = \inf_{\mu} \max_{a \in A} dist(a, \mu(a))$$

where $dist(a, b)$ denotes the Euclidean distance between point a and b and $\mu : A \to B$ is a continuous and non-decreasing function that maps every point in $a \in A$ to a point in $\mu(a) \in B$.

Since the early 90's the problem of computing the Fréchet distance between two polygonal curves has received considerable attention. In 1992 Alt and Godau [5] were the first to consider the problem and gave an $O(n^2 \log n)$ time algorithm for the problem. The only improvement since then is a randomized algorithm with running time $O(n^2(\log \log n)^2)$ in the word RAM model by Buchin *et al.* [12]. In 2014 Bringmann [9] showed that, conditional on the Strong Exponential Time Hypothesis (SETH), there cannot exist an algorithm with running time $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$. Even for realistic models of input curves, such as c-packed curves [21], exact distance computation requires $n^{2-o(1)}$ time under SETH [9]. Only by allowing a $(1+\varepsilon)$-approximation can one obtain near-linear running times in n and c on c-packed curves [10, 21].

For some applications, such as protein matching [33] and handwriting recognition [57], it is desirable to match the two curves under translation before computing the Fréchet distance between them. Formally, we match two polygonal curves A and B under the Fréchet distance by computing the translation τ so that the Fréchet distance is minimised. This variant is called the Translation Invariant Fréchet distance, and algorithms to compute it are well studied [6, 11, 33, 67]. Algorithms for the Translation Invariant Fréchet distance generally carry higher running times than for the standard Fréchet distance, moreover, these running times depend on the dimension of the input curves and whether the input curves are discrete or continuous.

For a discrete sequence of points in two dimensions, Bringmann *et al.* [11] recently provided an $\mathcal{O}(n^{4\frac{2}{3}})$ time algorithm to compute the Translation Invariant Fréchet distance, and showed that the problem has a conditional lower bound of $\Omega(n^4)$ under SETH. For continuous polygonal curves in two dimensions, Alt *et al.* [6] provided an $\mathcal{O}(n^8 \log n)$ time algorithm, and Wenk [67] extended this to an $\mathcal{O}(n^{11} \log n)$ time algorithm in three dimensions. If we allow for a $(1+\varepsilon)$-approximation then there is an $O(n^2/\varepsilon^2)$ time algorithm [6], which matches conditional lower bound for approximating the standard Fréchet distance [9].

For both the standard Fréchet distance and the Translation Invariant Fréchet distance, subquadratic and subquartic time algorithms respectively are unlikely to exist under SETH [9, 11]. However, if at least one of the trajectories can be preprocessed, then the Fréchet distance can be computed much more efficiently.

Querying the standard Fréchet distance between a given trajectory and a query trajectory has been studied [16, 18, 21, 31, 32], but due to the difficult nature of the query problem, data structures only exist for answering a restricted class of queries. There are two results which are most relevant. The first is De Berg *et al.*'s [18] data structure, which answers Fréchet distance queries between a horizontal query segment and a vertex-to-vertex subtrajectory of a preprocessed trajectory. Their data structure can be constructed in $O(n^2 \log^2 n)$ time using $O(n^2 \log^2 n)$ space such that queries can be answered in $O(\log^2 n)$ time. The second is Driemel and Har-Peled's [21] data structure, which answers approximate Fréchet distance queries between a query trajectory of complexity k and a vertex-to-vertex subtrajectory of a preprocessed trajectory. The data structure can be constructed in $\mathcal{O}(n \log^3 n)$ using $\mathcal{O}(n \log n)$ space, and a constant factor approximation to the Fréchet distance can be answered in $\mathcal{O}(k^2 \log n \log(k \log n))$ time. In the special case when $k = 1$, the approximation ratio can be improved to $(1+\varepsilon)$ with no increase in preprocessing or query time with respect to n. New ideas are required for exact Fréchet distance queries on arbitrary query trajectories. Other query versions for the standard Fréchet distance have also been considered [16, 31, 32].

Querying the Translation Invariant Fréchet distance is less well understood.

This is not surprising given the complexity of computing the Translation Invariant Fréchet distance. Nevertheless, in our paper we are able to answer exact Translation Invariant Fréchet queries in a restricted setting of horizontal query segments. We hope this will be a step towards answering exact Translation Invariant Fréchet queries between arbitrary trajectories.

In this paper, we answer exact Translation Invariant Fréchet distance queries between a subtrajectory (not necessarily vertex-to-vertex) of a preprocessed trajectory and a horizontal query segment. The data structure can be constructed in $O(n^2 \log^2 n)$ time using $O(n^2 \log^2 n)$ space such that queries can be answered in $O(\text{polylog}\, n)$ time. We use Megiddo's parametric search technique [39] on De Berg *et al.*'s [18] data structure to optimise the Fréchet distance. We hope that as standard Fréchet distance queries become more well understood, similar optimisation methods could lead to improved data structures for the Translation Invariant Fréchet distance as well.

3.1 Preliminaries

Let $p_1, \ldots, p_n$ be a sequence of n points in the plane. We denote $\pi = (p_1, p_2 \ldots, p_n)$ to be the polygonal trajectory defined by this sequence. Let $x_0 \leq x_1$ and $y \in \mathbb{R}$, and define $p = (x_0, y)$ and $q = (x_1, y)$ so that $Q = pq$ is a horizontal segment in the plane. Let u and v be two points on the trajectory π, then from [18], the Fréchet distance between $\pi[u, v]$ and Q can be computed by using the formula:

$$\delta_F(\pi[u, v], pq) = \max\{\|up\|, \|vq\|, \delta_{\overrightarrow{h}}(\pi[u, v], pq), B(\pi[u, v], y)\}.$$

The first two terms are simply the distance between the starting points of the two trajectories, and the ending points of the two trajectories. The third term is the directed Hausdorff distance between $\pi[u, v]$ and Q which can be computed from:

$$\delta_{\overrightarrow{h}}(\pi[u, v], Q) = \max\{\max_{p_i.x \in (-\infty, x_0]} \|p - p_i\|, \max_{p_i.x \in [x_1, \infty)} \|q - p_i\|, \max_i \|y - p_i.y\|\},$$

where each p_i in the formula above are vertices of the subtrajectory $\pi[u, v]$, and $p_i.x$ are their x-coordinates. The formula handles three cases for mapping every point of $\pi[u, v]$ to its closest point on Q. The first term describes mapping points of $\pi[u, v]$ to the left of p to their closest point p. The second term describes mapping points of $\pi[u, v]$ to the right of q analogously. The third term describes mapping points of $\pi[u, v]$ that are in the vertical strip between p and q to their orthogonal projection onto Q. In later sections we refer to these three terms as $\delta_{\overrightarrow{h}}(L)$, $\delta_{\overrightarrow{h}}(R)$ and $\delta_{\overrightarrow{h}}(M)$ for the left, right, and middle terms of the Hausdorff distance respectively.

The fourth term in our formula for the Fréchet distance is the maximum backward pair distance over all backward pairs. A pair of vertices (p_i, p_j) (with $j > i$) is a backward pair if p_j lies to the left of p_i. The backward pair distance of $\pi[u, v]$ can be computed from:

$$B(\pi[u, v], y) = \max_{\forall p_i, p_j \in \pi[u,v]: i \leq j, p_i.x \geq p_j.x} B_{(p_i, p_j)}(y),$$

where $B_{(p_i,p_j)}(y)$ is the backward pair distance for a given backward pair (p_i, p_j) and is defined as

$$B_{(p_i,p_j)}(y) = \min_{x \in \mathbb{R}} \max\{\|p_i - (x, y)\|, \|p_j - (x, y)\|\}.$$

The distance terms in the braces compute the distance between a given point (x, y) and the farthest of p_i and p_j. Let us call this the backward pair distance of (x, y). Then the function $B_{(p_i,p_j)}(y)$ denotes the minimum backward pair distance of a given backward pair (p_i, p_j) over all points (x, y) which have the same y-coordinate. Taking the maximum over all backward pairs gives us the backward pair distance for $\pi[u, v]$.

In Figure 3.1, we show for each y-coordinate the point with the minimum backward pair distance (left), and the magnitude of this minimum distance (right). We see in the figure that the function $B_{(p_i,p_j)}(y)$ consists of two linear functions joined together in the middle with a hyperbolic function.

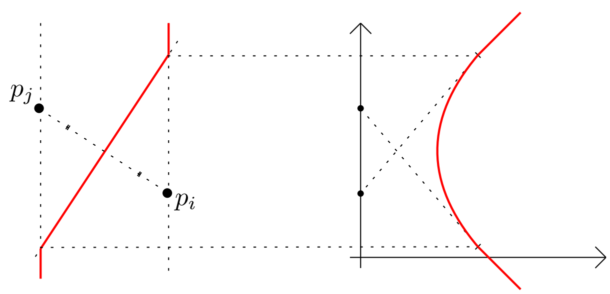


Figure 3.1: For each y-coordinate, Left: the point with minimum backward pair distance, Right: the minimum backward pair distance.

We extend the work of De Berg *et al.* [18] in two ways. First, we provide a method for answering Fréchet distance queries between $\pi[u, v]$ and Q when u and v are not necessarily vertices of π, and second, we optimise the placement of Q to minimise its Fréchet distance to $\pi[u, v]$. We achieve both of these extensions by carefully applying Megiddo's parametric search technique [39] to compute the optimal Fréchet distance.

In order to apply parametric search, we are required to construct a set of critical values (which we will describe in detail at a later stage) so that an optimal solution is guaranteed to be contained within this set. Since this set of critical values is often large, we need to avoid computing the set explicitly, but instead design a decision algorithm that efficiently searches the set implicitly. Megiddo's parametric search [39] states that if:

- the set of critical values has polynomial size, and

- the Fréchet distance is convex with respect to the set of critical values, and
- a comparison-based decision algorithm decides if a given critical value is equal to, to the left of, or to the right of the optimum,

then there is an efficient algorithm to compute the optimal Fréchet distance in $\mathcal{O}(PT_p + T_pT_s \log P)$ time, where P is the number of processors of the (parallel) algorithm, T_p is the parallel running time and T_s is the serial running time of the decision algorithm. For our purposes, $P = 1$ since we run our queries serially, and $T_p = T_s = \mathcal{O}(\text{polylog}\, n)$ for the decision versions of our query algorithms.

3.2 Computing the Fréchet Distance

We preprocess π into a data structure such that for a query specified by:

1. two points u and v on the trajectory π (not necessarily vertices),
2. a horizontal segment Q,

we can quickly compute the exact Fréchet distance between Q and the subtrajectory $\pi[u, v]$.

To achieve such a data structure, we first define the following notation. Let p_u be the first vertex of π along $\pi[u, v]$ and let p_v be the last vertex of π along $\pi[u, v]$, as illustrated in Figure 3.2.

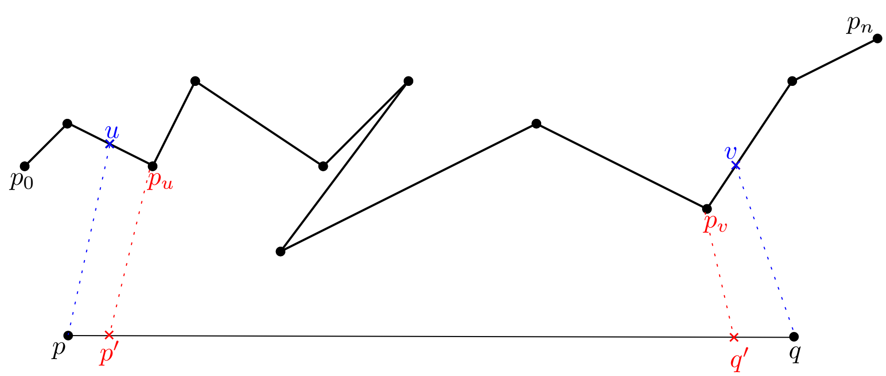


Figure 3.2: The points p' and q' mapped to the vertices p_u and p_v of the trajectory.

If p_u and p_v do not exist then $\pi[u, v]$ is a single segment so the Fréchet distance between $\pi[u, v]$ and Q can be computed in constant time. Otherwise, our goal is to build a Fréchet mapping $\mu : \pi[u, v] \to Q$ which attains the optimal Fréchet distance. We build this mapping μ in several steps. Our first step is to compute points p' and q' on the horizontal segment pq so that $p' = \mu(p_u)$ and $q' = \mu(p_v)$.

If the point p' is computed correctly, then the mapping $p' \to p_u$ allows us to subdivide the Fréchet computation into two parts without affecting the overall value of the Fréchet distance. In other words, we obtain the following formula:

$$\delta_F(\pi[u,v], pq) = \max\{\delta_F(up_u, pp'),\ \delta_F(\pi[p_u,v], p'q)\} \tag{3.1}$$

We now apply the same argument to p_v. We compute q' optimally on the horizontal segment $p'q$ optimally so that mapping $p_v \to q'$ does not increase the Fréchet distance between the subtrajectory $\pi[p_u,v]$ and the truncated segment $p'q$. In other words, we have:

$$\delta_F(\pi[u,v], pq) = \max\{\delta_F(up_u, pp'),\ \delta_F(\pi[p_u,p_v], p'q'),\ \delta_F(p_v v, q'q)\} \tag{3.2}$$

Now that p_u and p_v are vertices of π, [18] provides an efficient data structure for computing the middle term $\delta_F(\pi[p_u,p_v], p'q')$. The first and last terms have constant complexity and can be handled in constant time. All that remains is to compute the points p' and q' efficiently.

Theorem 1. Given a trajectory π with n vertices in the plane. There is a data structure that uses $\mathcal{O}(n^2 \log^2 n)$ space and preprocessing time, such that for any two points u and v on π (not necessarily vertices of π) and any horizontal query segment Q in the plane, one can determine the exact Fréchet distance between Q and the subtrajectory from u to v in $\mathcal{O}(\log^8 n)$ time.

Proof. **Decision Algorithm.** Let S be the set of critical values (defined later in this proof), let s be the current candidate for the point p', and let $F(s) = \max(\delta_F(ps, up_u), \delta_F(sq, \pi[p_u,v]))$ be the minimum Fréchet distance between pq and $\pi[u,v]$ subject to p_u being mapped to s. Our aim is to design a decision algorithm that runs in $O(\log^4 n)$ time that decides whether the optimal p' is equal to s, to the left of s or to the right of s. This is equivalent to proving that all points to one side of s cannot be the optimal p' and may be discarded.

We use the Fréchet distance formula from Section 3.1 to rewrite $F(s) = \max(\|up\|, \|vq\|,\ \|p_u s\|,\ \delta_{\overrightarrow{h}}(\pi[p_u,v], sq), B(\pi[p_u,v], y))$. Then we take several cases for which of these five terms attains the maximum value $F(s)$, and in each case we either deduce that $p' = s$ or all critical values to one side of s may be discarded.

- If $F(s) = \max(\|up\|, \|vq\|, B(\pi[p_u,v], y))$, then $p' = s$. We observe that none of the three terms on the right hand side of the equation depend on the position of s. Hence, $F(s) = \max(\|up\|, \|vq\|, B(\pi[p_u,v], y)) \le F(p')$, and since $F(p')$ is the minimum possible value, $F(s) = F(p')$. We have found a valid candidate for p' and can discard all other candidates in the set S.

- If $F(s) = \|p_u s\|$ and p_u is to the right (left) of s, then p' is to the right (left) of s. We will argue this for when p_u is to the right of s, but an analogous argument can be used when p_u is to the left. We observe that all points t to the left of s will now have $\|p_u t\| > \|p_u s\|$. Hence, $F(s) = \|p_u s\| < \|p_u t\| \le F(t)$ for all points t to the left of s, therefore all points to the left of s may be discarded.

- If $F(s) = \delta_{\overrightarrow{h}}(\pi[p_u, v], sq)$, then p' is to the left of s. The directed Hausdorff distance maps every point in $\pi[p_u, v]$ to their closest point on sq, so by shortening sq to tq for some point t on sq to the right of s, the directed Hausdorff distance cannot decrease. Hence, $F(s) \leq F(t)$ for all t to the right of s, so all points to the right of s may be discarded.

To determine q' for a fixed candidate s for p', we treat the problem in a similar way. We consider the subtrajectory $\pi[p_u, v]$ and the horizontal line segment sq. Defining a function $G(t)$ representing the Fréchet distance when p_v is mapped to t, we obtain a similar decision algorithm. The most notable difference is that since we now consider the end of the subtrajectory, the decisions for moving t left and right are reversed.

Convexity. We will prove that $F(s)$ is convex, and it will follow similarly that $G(t)$ is convex. It suffices to show that $F(s)$ is the maximum of convex functions, since the maximum of convex functions is itself convex. The three terms $\|up\|$, $\|vq\|$, $B(\pi[p_u, v], y)$ are constant. The term $\|p_u s\|$ is an upward hyperbola and is convex. If suffices to show that $\delta_{\overrightarrow{h}}(\pi[p_u, v], sq)$ is convex.

We observe that the Hausdorff distance $\delta_{\overrightarrow{h}}(\pi[p_u, v], sq)$ must be attained at a vertex p_i of $\pi[p_u, v]$, and that each of $\delta_{\overrightarrow{h}}(p_i, sq)$ as a function of s is a constant function between p and p_i^*, and a hyperbolic function between p_i^* and q. Thus, the function for each p_i is convex, so the overall Hausdorff distance function is also convex.

Critical Values. A critical value is a value c which could feasibly attain the minimum value $F(c) = F(p')$. We represent $F(s)$ as the minimum of n simple functions and then argue that the minimum of F can only occur at the minimum of one of these functions, or at the intersection of a pair of these functions.

First, $\|up\|, \|vq\|, B(\pi[p_u, v], y)$ are constant functions in terms of s. Next, $\|p_u s\|$ is a hyperbolic function. Finally, $\delta_{\overrightarrow{h}}(\pi[p_u, v], sq)$ is not itself simple, but it can be rewritten as the combination of n simple functions as described in the above section.

Hence, $F(s)$ is the combination (maximum) of n simple functions, and these functions are simple in that they are piecewise constant or hyperbolic. Hence $F(s)$ attains its minimum either at the minimum of one of these n functions, or at a point where two of these functions intersect. Therefore, there are at most $\mathcal{O}(n^2)$ critical values for $F(s)$.

Query Complexity. Computing q' for a given candidate s for p' takes $\mathcal{O}(\log^4 n)$ time: We can compute the terms $\|up\|$, $\|p_u s\|$, $\|vq\|$, and $\|p_v q'\|$ in constant time. The terms $B(\pi[p_u, p_v], y)$ and $\delta_{\overrightarrow{h}}(\pi[p_u, p_v], sq')$ can be computed in $\mathcal{O}(\log^2 n)$ time using the existing data structure by De Berg *et al.* [18]. We need to determine the time complexity of the sequential algorithm T_s, parallel algorithm T_p, and the number of the processor P. To find q', the decision algorithm takes $T_s = \mathcal{O}(\log^2 n)$. The parallel form runs on one processor in $T_p = \mathcal{O}(\log^2 n)$. Substituting these values in the running time of the parametric search of $\mathcal{O}(PT_p + T_p T_s \log P)$ leads to $\mathcal{O}(\log^4 n)$ time.

The above analysis implies that p' itself can be computed in $\mathcal{O}(\log^8 n)$ time: For a given s, the decision algorithm runs in $T_s = \mathcal{O}(\log^4 n)$ as mentioned above. The parallel form of the decision algorithm runs on one processors in $T_p = \mathcal{O}(\log^4 n)$. Substituting these values in the running time of the parametric search of $\mathcal{O}(PT_p + T_pT_s \log P)$ leads to $\mathcal{O}(\log^8 n)$ time.

Preprocessing and Space. To compute the second term of Formula 3.2, we use the data structure by De Berg *et al.* [18]. This data structure uses $\mathcal{O}(n^2 \log^2 n)$ space and preprocessing time and supports $\mathcal{O}(\log^2 n)$ query time. □

We note that the set of critical values can be restricted significantly, while still being guaranteed to contain optimal elements to use as p' and q'. Specifically, in Subsection 3.2.1, we show that we can reduce the size of this set from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

3.2.1 Improving the Number of Critical Values

In this section we improve the number of critical values for p' and q'. Note that our parametric search does an implicit search over these critical values, therefore, only a logarithmic number of critical values are ever explicitly computed. For this reason, reducing the number of critical values for p' and q' does not affect the running time of the algorithm in Theorem 1.

Recall that our query consists of two points u and v on the trajectory π, as well as a horizontal segment with endpoints p and q. Recall also that p_u and p_v are the first and last vertices of π on the subtrajectory $\pi[u, v]$. As we saw in Theorem 1, computing the Fréchet distance reduces down to computing p' and q' such that there exists a mapping $\mu(p_u) = p'$ and $\mu(p_v) = q'$. In this case we say that p' represents $\mu(p_u)$. Formally, we have:

Definition 1. A point s represents p_u if and only if there exists a non-decreasing continuous mapping $\mu : \pi[u, v] \to pq$ such that μ achieves the Fréchet distance and $\mu(p_u) = s$.

Hence, to improve the number of critical values and get the stated bound, we need to show for any query points u and v on π and any horizontal segment pq in the plane, that there are at most $O(n)$ points on pq which could represent p_u. Now we define a collection of points on pq that could feasibly be representatives.

Definition 2. Given any vertex p_i on the subtrajectory $\pi[u, v]$, let p_i^* be the orthogonal projection of vertex p_i onto the horizontal segment pq.

Definition 3. Given any two vertices p_i and p_j on the subtrajectory $\pi[u, v]$, let P_{ij} be the perpendicular bisector of p_i and p_j. Let P_{ij}^* be the intersection of the perpendicular bisector P_{ij} with the horizontal segment pq.

We now have all we need in place to define our set S of candidates for p' and q'.

Definition 4. Let S be the set containing the following elements:

1. the points p and q,

2. all orthogonal projection points p_i^*, and
3. all perpendicular bisector intersection points P_{ij}^*.

It now suffices to show that S contains at least one representative for p_u. An analogous argument shows that S contains a representative of p_v as well.

Lemma 1. There exists an element $s \in S$ on pq that represents p_u.

Proof. Assume for the sake of contradiction that there is no element $s \in S$ which represents p_u. Consider a mapping μ that achieves the Fréchet distance and consider the point $\mu(p_u)$ on the horizontal segment pq. Since $\mu(p_u)$ represents p_u, $\mu(p_u)$ cannot be in S and must lie strictly between two consecutive elements of S, say s_L to its left and s_R to its right (see Figure 3.3). Note that it may be the case that $s_L = p$ or $s_R = q$. Since s_L and s_R are elements of S, neither can represent p_u. Next, we reason about the implications of s_L and s_R not being able to represent p_u, before putting these together to obtain a contradiction.

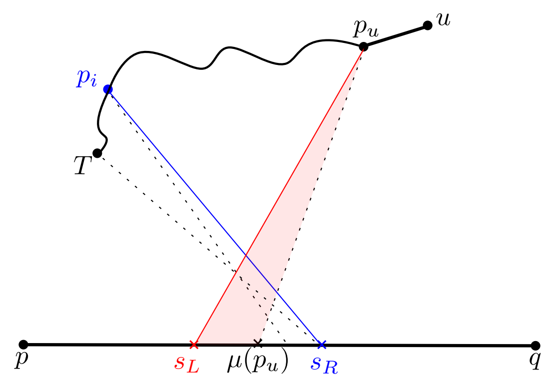


Figure 3.3: The point $\mu(p_u)$ lies between two consecutive elements s_L and s_R. Distances that are greater than d are thin solid and distances that are at most d are dotted, where d is the Fréchet distance.

***s_L* cannot represent *p_u*.** This means that no mapping which sends $p_u \to s_L$ achieves the Fréchet distance. Let us take the mapping μ and modify it into a new mapping μ_L in such a way that $\mu_L(p_u) = s_L$. We can do so by starting out parametrising μ_L with a constant speed mapping which sends $u \to p$ and $p_u \to s_L$. Next, we stay fixed at p_u along the subtrajectory and move along the horizontal segment from s_L to $\mu(p_u)$. The red shaded region in Figure 3.3 describes this portion of the remapping. Now that $\mu_L(p_u) = \mu(p_u)$, we can use the original mapping for the rest.

Since our new mapping μ_L maps p_u to an element of S that cannot represent it, we know that our modification must increase the Fréchet distance. The only place where the Fréchet distance could have increased is at the line segments where the mapping was changed and here $\mu_L(p_u) = s_L$ maximises the Fréchet distance. Hence, we have $\|p_u s_L\| > d$, where d is the Fréchet distance, as shown in Figure 3.3. But $\|p_u\mu(p_u)\| \le d$, so we can deduce that p_u is closer to $\mu(p_u)$ than s_L. Therefore, p_u is to the right of s_L. Finally, if s_L and s_R were on opposite sides of p_u^*, then s_L and s_R would not be consecutive, therefore p_u must be on the same side of s_L and s_R. Therefore, p_u is to the right of the entire segment s_Ls_R.

$\boldsymbol{s_R}$ **cannot represent** $\boldsymbol{p_u}$**.** Again, no mapping which sends $p_u \to s_R$ achieves the Fréchet distance, so we use the same approach and modify μ into a new mapping mapping μ_R in such a way that $\mu_R(p_u) = s_R$. To this end, we keep the mapping μ_R the same as μ until it reaches p_u, and then while staying at p_u, we fastforward the movement from $\mu(p_u)$ along the horizontal segment so that $\mu_R(p_u) = s_R$. Next, we stay at s_R and fastforward the movement along the subtrajectory, until we reach the first point T on the subtrajectory such that $\mu(T) = s_R$ in the original mapping. From point T onwards we can use the original mapping μ.

Since our new mapping μ_R maps p_u to an element of S that does not represent it, we cannot have achieved the Fréchet distance. The first change we applied was staying at p_u and fastforwarding the movement from $\mu(p_u)$ to s_R. However, since we know from above that p_u is to the right of the entire segment s_Ls_R, this fastforwarding moves closer to p_u, so this part cannot increase the Fréchet distance. The second change we applied, staying at s_R and fastforwarding the movement from p_u to T, must therefore be the change that increases the Fréchet distance. Thus, there must be a point on the subtrajectory $\pi[p_u, T]$ which has distance greater than d, the Fréchet distance, to the point s_R. Since the distance to a point s_R is maximal at vertices of $\pi[p_u, T]$, we can assume without loss of generality that $\|p_is_R\| > d$ for some vertex p_i. Consider $\mu(p_i)$ in the original mapping. Since p_i is on the subtrajectory $\pi[p_u, T]$, $\mu(p_i)$ must be between $\mu(p_u)$ and $\mu(T) = s_R$. This mapping of p_i to $\mu(p_i)$ is shown as a black dotted line in Figure 3.3. Using a similar logic as before, $\|p_i\mu(p_i)\| \le d$ and $\|p_is_R\| > d$, so p_i must lie to left of s_R. And since s_L and s_R are consecutive elements of S, we deduce that p_i is to the left of the entire segment s_Ls_R.

Putting these together. We now have the full diagram as shown in Figure 3.3. The vertex p_u is to the right of both s_L and s_R and the vertex p_i is to the left of both s_L and s_R. We also have inferred that $\|p_us_L\| > d$ and $\|p_is_R\| > d$. Moreover, since $\|p_u\mu(p_u)\| \le d$ and $\|p_i\mu(p_i)\| \le d$, we also have that $\|p_us_R\| \le d$ and $\|p_is_L\| \le d$, since this just moves these endpoints closer to p_u and p_i respectively.

Finally, we will show that P_{ui}^* lies between s_L and s_R, reaching the intended contradiction. We do so by considering the function $f(x) = \|xp_u\| - \|xp_i\|$ for all points x between s_L and s_R. From our length conditions, we have that $f(s_L) > 0$, $f(s_R) < 0$. Furthermore, since $f(x)$ is a continuous function, by the intermediate value theorem, there is a point x strictly between s_L and s_R such that $f(x) = 0$. Since $f(x) = 0$, the point x is equidistant from p_u and p_i so therefore lies on both P_{ui} and the horizontal segment pq. Therefore $x = P_{ui}^*$ and is an element of S between two consecutive elements s_L and s_R, giving us a contradiction. $\square$

Note that in the above proof, we require only P^*_{ui} to be in the candidate set when we are computing p', and also only when (p_u, p_i) is a backward pair. Recall from Section 3.1 and [17] that (p_u, p_i) is a backward pair if p_i is after p_u along π and p_i lies to the left of p_u. This means that for computing p' and q' respectively, we only require the bisector intersections P^*_{ui} and P^*_{jv} to be in S, of which there are $O(n)$, and therefore the size of S is at most $\mathcal{O}(n)$.

3.3 Minimizing the Fréchet Distance Under Vertical Translation

We preprocess π into a data structure such that for a query specified by:

1. two points u and v on the trajectory π,
2. two vertical lines x_1 and x_2 such that $\|x_2 - x_1\| = L$,

we can quickly find a horizontal segment l_y that spans the vertical strip between x_1 and x_2 such that the Fréchet distance between l_y and the subtrajectory $\pi[u, v]$ is minimised; see Figure 3.4.

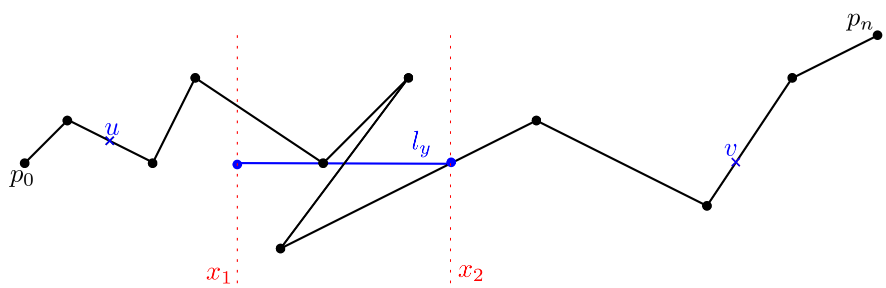


Figure 3.4: Finding a horizontal segment l_y in the vertical strip between x_1 and x_2 that minimises the Fréchet distance between l_y and $\pi[u, v]$.

In other words, we focus on a special case where the horizontal segment can only be translated vertically. In Section 3.4 we consider arbitrary translations of the horizontal segment.

In the next theorem, we present a decision problem $D_{\pi[u,v]}(x_1, x_2, l^c_y)$ that, for a given trajectory π with two points u and v on π and two vertical lines $x = x_1$ and $x = x_2$, returns whether the line l_y is above, below, or equal to the current candidate line l^c_y. We then use parametric search to find l_y that minimises the Fréchet distance.

Theorem 2. Given a trajectory π with n vertices in the plane. There is a data structure that uses $\mathcal{O}(n^2 \log^2 n)$ space and preprocessing time, such that for any two points u and v on π (not necessarily vertices of π) and two vertical lines $x = x_1$ and $x = x_2$, one can determine the horizontal segment l_y with left endpoint on $x = x_1$ and right endpoint on $x = x_2$ that minimises its Fréchet distance to the subtrajectory $\pi[u, v]$ in $\mathcal{O}(\log^{16} n)$ time.

Proof. **Decision Algorithm.** Let l_y^c be the current horizontal segment. To decide whether the line segment that minimises the Fréchet distance lies above or below l_y^c, we must compute the maximum of the terms that determine the Fréchet distance: $\|up\|$, $\|vq\|$, $\delta_{\overrightarrow{h}}(\pi[u, v], pq)$, and $B(\pi[u, v], l_y^c)$. As mentioned in Section 3.1, we divide the directed Hausdorff distance into three different terms: $\delta_{\overrightarrow{h}}(L)$, $\delta_{\overrightarrow{h}}(R)$, and $\delta_{\overrightarrow{h}}(M)$. We first consider when one term determines the Fréchet distance, in which we have the following cases:

- $\|up\|$, $\|vq\|$, $\delta_{\overrightarrow{h}}(L)$, and $\delta_{\overrightarrow{h}}(R)$: Since the argument for these terms is analogous, we focus on $\|up\|$. If u is located above l_y^c, the next candidate lies above l_y^c (search continues above l_y^c). If u lies below l_y^c, the next candidate lies below l_y^c (search continues below l_y^c). If u and p have the same y-coordinate, we can stop, since moving l_y^c either up or down increases the Fréchet distance.
- $B(\pi[u, v], l_y^c)$: If the midpoint of the perpendicular bisector of the backward pair determining the current Fréchet distance is located above l_y^c, the next candidate lies above l_y^c, since this is the only way to decrease the distance to the further of the two points of the backward pair. If this midpoint lies below l_y^c, the next candidate lies below l_y^c. If the midpoint is located on l_y^c, we can stop, because the term $B_{(p_i, p_j)}(l_y^c)$ increases by either moving l_y^c up or down.
- $\delta_{\overrightarrow{h}}(M)$: If the point with maximum projected distance is located above l_y^c, the next candidate lies above l_y^c. If the point is below l_y^c, the next candidate lies below l_y^c. If the point is on l_y^c, then we stop, but unlike in the first case, this maximum term and the overall Fréchet distance must both be zero in this case.

If more than one term determine the current Fréchet distance, we must first determine the direction of the implied movement for each term. If this direction is the same, we move in that direction. If the directions are opposite, we can stop, because moving in either direction would increase the other maximum term resulting in a larger Fréchet distance.

Convexity. It suffices to show the Fréchet distance between $\pi[u, v]$ and l_y^c as a function of y is convex. We show that this function is the maximum of several convex functions, and therefore must be convex. The first two terms for computing the Fréchet distance are $\|up\|$ and $\|vq\|$, which are hyperbolic in terms of y. Similarly to the previous section, we handle each of the Hausdorff distances by splitting them up Hausdorff distances for each vertex p_i. The left and right Hausdorff distances $\delta_{\overrightarrow{h}}(L)$ and $\delta_{\overrightarrow{h}}(R)$ for a single vertex p_i is a hyperbolic function. The middle Hausdorff distance $\delta_{\overrightarrow{h}}(M)$ for a single vertex p_i is a shifted absolute value function. In all cases, Hausdorff distance for a single vertex is convex, so the overall Hausdorff distance is also convex. Finally, the backward pair distance $B(\pi[u,v], l_y^c)$ as a function of y is shown by De Berg *et al.* [18] to be two rays joined together in the middle with a hyperbolic arc. It is easy to verify that this function is convex.

Critical Values. A horizontal segment l_y^c is a critical value of a decision algorithm if the decision algorithm could feasibly return that $l_y^c = l_y$. These critical values are the y-coordinates of the intersection points of two hyperbolic functions for each combination of two terms of determining the Fréchet distance or the minimum point of the upper envelope of two such hyperbolic functions. Therefore, there are only a constant number of critical values for each two terms. Each term gives rise to $\mathcal{O}(n^2)$ hyperbolic functions (specifically, $B(\pi[u,v], l_y^c)$ can be of size $\Theta(n^2)$ in the worst case). Thus, there are $\mathcal{O}(n^4)$ critical values.

Query Complexity. The decision algorithm runs in $T_s = T_p = \mathcal{O}(\log^8 n)$ time since we use Theorem 1 to compute the Fréchet distance for a fixed l_y^c. Substituting this in the running time of the parametric search $\mathcal{O}(PT_p + T_pT_s \log P)$ leads to a query time of $\mathcal{O}(\log^{16} n)$.

Preprocessing and Space. Since we compute the Fréchet distance of the current candidate l_y^c using Theorem 1, we require $\mathcal{O}(n^2 \log^2 n)$ space and preprocessing time. □

3.4 Minimizing the Fréchet Distance for Arbitrary Placement

We preprocess π into a data structure such that for a query specified by:

1. two points u and v on the trajectory π,
2. a positive real value L,

we can quickly determine the horizontal segment l of length L such that the Fréchet distance between l and the subtrajectory $\pi[u,v]$ is minimised.

In the following theorem, we present a decision problem $D_{\pi[u,v]}(L, x_1)$ that, for a given trajectory π with two points u and v on π and a length L and an x-coordinate x_1, returns whether the line l has its left endpoint to the left, on, or to the right of x_1. We then apply parametric search to this decision algorithm to find the horizontal segment l of length L with minimum Fréchet distance to $\pi[u,v]$.

Theorem 3. Given a trajectory π with n vertices in the plane. There is a data structure that uses $\mathcal{O}(n^2 \log^2 n)$ space and preprocessing time, such that for any two points u and v on π (not necessarily vertices of π) and a length L, one can determine the horizontal segment l of length L that minimises the Fréchet distance to $\pi[u,v]$ in $\mathcal{O}(\log^{32} n)$ time.

Proof. **Decision Algorithm.** We only need to decide whether l^c should be moved to the left or right, with respect to its current position, for the cases where $D_{\pi[u,v]}(x_1, x_2, l_y)$ stops. We classify the terms that determine the Fréchet distance in two classes:

- C_1: This class contains the terms whose value is determined by the distance from a point on $\pi(u, v)$ to p or q. Hence, it consists of $\|up\|$, $\|vq\|$, $\delta_{\overrightarrow{h}}(R)$, and $\delta_{\overrightarrow{h}}(L)$.
- C_2: This class contains the terms whose value is determined by the distance from a point on $\pi(u, v)$ to the closest point on pq. Hence, it consists of $\delta_{\overrightarrow{h}}(M)$ and $B(\pi[u, v], l_y)$.

Next, we show how to decide whether the next candidate line segment lies to the left or right of l^c (i.e., the x-coordinate of its left endpoint lies to the left or right of the left endpoint of l^c) for each case where $D_{\pi[u,v]}(x_1, x_2, l_y)$ stops.

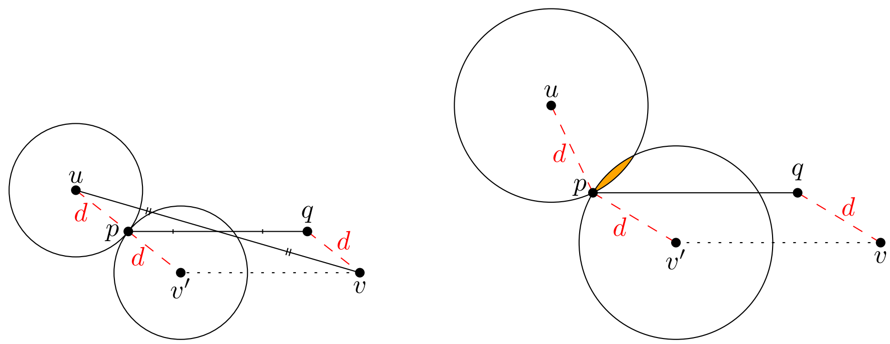
We decide this by considering each C_1 and C_2 term and the restriction they place on the next candidate line segment pq. After we do this for each individual C_1 or C_2 term, we take the intersection of all these restrictions. If the intersection is empty, then our placement of pq was optimal, and our decision algorithm stops. Otherwise we can either move pq to the left or to the right to improve the Fréchet distance.

First, consider the C_1 terms. Let us assume for now that the C_1 term is the distance term $\|up\|$. Then in order to improve the Fréchet distance to u, we need to place the horizontal segment pq in such a way that p lies inside the *open* disk centered at u with radius equal to the current Fréchet distance d. A similar condition holds for the other C_1 terms: each defines a disk of radius d and the point it maps to in the next candidate needs to lie inside this disk.

Similarly, the C_2 terms define horizontal open half-planes. Consider the term $\delta_{\overrightarrow{h}}(M)$. This term is reduced when the vertical projection distance to the line segment is reduced. Hence, if the point defining this term lies above l^c, this term can be reduced by moving the line segment upward and thus the half-plane is the half-plane above l^c. An analogous statement holds if the point lies below l^c. For the term $B(\pi[u, v], l_y)$, we need to consider the midpoint of the bisector, since the implied Fréchet distance is the distance from l^c to the further of the two points defining the bisector. Thus, the half-plane that improves the Fréchet distance is the one that lies on the same side of l^c as this midpoint.

To combine all the terms we do the following: First, we take all disks induced by the C_1 terms whose distance is with respect to q and translate them horizontally to the left by a distance of L. This ensures that the disks constructed with respect to p can now be intersected with the disks constructed with respect to q. We take the intersection of all C_1 and C_2 terms that defined the stopping condition of the vertical optimisation step. If this intersection is empty, by construction there is no point where we can move p to in order to reduce the Fréchet distance. If it is not empty, we will show that it lies entirely to the left or entirely to the right of p and thus implies the direction in which the next candidate lies.

Now that we have described our general approach, we show which cases can occur and show that for each of them we can determine in which direction to continue (if any).



(a) The midpoint of pq is the midpoint of uv.

(b) Moving the midpoint of pq towards the midpoint of uv.

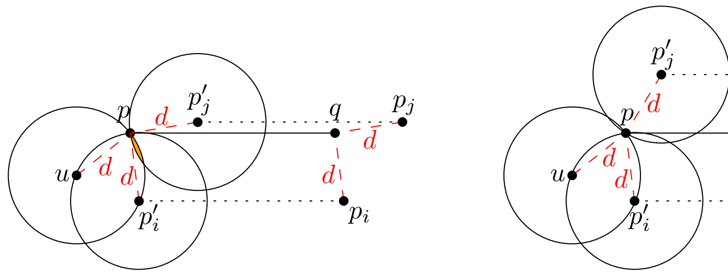
Figure 3.5: Determining where l^c should be moved to reduce the Fréchet distance.

Case 1. $D_{\pi[u,v]}(x_1, x_2, l_y)$ stops because of terms in C_1. If only a single term of C_1 is involved, say $\|up\|$, this implies that the y-coordinate of u is the same as that of l^c and thus its disk lies entirely to the left of p. Hence, we can reduce the Fréchet distance by moving l^c horizontally towards u and thus we pick our next candidate in that direction. The same argument follows analogously the C_1 term is $\|vq\|$, $\delta_{\overrightarrow{h}}(R)$, or $\delta_{\overrightarrow{h}}(L)$, the same argument follows analogously the distance is between a point on the trajectory

If two terms of C_1 are involved, say $\|up\|$ and $\|vq\|$, their intersection can be empty (see Figure 3.5(a)) or non-empty (see Figure 3.5(b)). If it is empty, the midpoint of pq is the same as the midpoint of uv, which implies that we cannot reduce the Fréchet distance. If the intersection is not empty, moving the endpoint of the line segment into this region potentially reduces the Fréchet distance. We note that since $\|up\|$ and $\|vq\|$ stopped the vertical optimisation, they lie on opposite sides of l^c. Hence, the intersection of their disks lies entirely to the left or entirely to the right of p and thus determines in which direction the next candidate lies.

If three terms in C_1 are involved, we again construct the intersection as described earlier. If this intersection is empty (see Figure 3.6b), we are again done. If it is not (see Figure 3.6a), it again determines the direction in which the our next candidate lies, as the intersection of three disks is a subset of the intersection of two disks.

If there are more than three C_1 terms, we reduce this to the case of three C_1 terms. If the intersection of these disks is non-empty, then trivially the intersection of a subset of three of them is also non-empty. If the intersection is empty, we select a subset of three whose intersection is also empty. The three disks can be chosen as follows. Insert the disks in some order and stop when the intersection first becomes empty. The set of three disks consists of the last



(a) The case where we can improve the Fréchet distance.

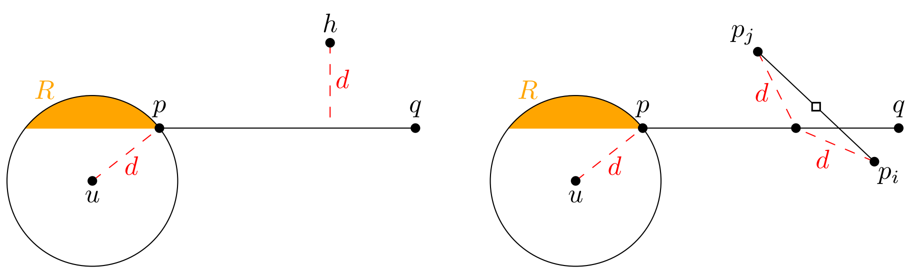
(b) The case where we cannot improve the Fréchet distance.

Figure 3.6: The case where we have three C_1 terms.

inserted disk and the two extreme disks among the previously inserted disks. Since the boundary of all the disks must go through a single point and the disks have equal radius, these three disks will have an empty intersection. Hence, the case of more than three disks reduces to the case of three disks.

Case 2. $D_{\pi[u,v]}(x_1, x_2, l_y)$ stops because of a term in C_2. Since the vertical optimisation stopped, we know that at least two C_2 terms are involved and there exists a pair that lies on opposite sides of l^c. These two terms define open half-planes whose intersection is empty, hence we cannot reduce the Fréchet distance further.

Case 3. $D_{\pi[u,v]}(x_1, x_2, l_y)$ stops because of a term in C_1 and a term in C_2. We can assume there are at most two C_1 terms and at most one C_2 terms, due to the previous cases.



(a) The case where the C_2 term is $\delta_{\overrightarrow{h}}(M)$.

(b) The case where the C_2 term is $\max\limits_{u \le i \le j \le v, p_i.x \ge p_j.x} B_{(p_i, p_j)}(l^c)$.

Figure 3.7: Reduce the Fréchet distance when it is determined by a term of C_1 and a term of C_2.

The C_2 term can be either $\delta_{\overrightarrow{h}}(M)$ (see Figure 3.7a, where h is the point at distance d) or $B(\pi[u, v], l_y)$ (see Figure 3.7b, where (p_i, p_j) is the backward pair with distance d). The region R shows the intersection of the disk of a single C_1 term and the C_2 term. We note that since the point of the C_1 term and the point of the C_2 term lie on opposite sides of l^c, this intersection lies either entirely to the left or entirely to the right of p or q, determining the direction in which our next candidate must lie.

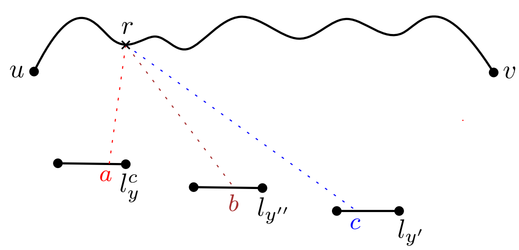
The same procedure can be applied when there are two C_1 terms and using similar arguments, it can be shown that if the intersection is not empty, the direction to improve the Fréchet distance is uniquely determined.

Convexity. Next, we show that $D_{\pi[u,v]}(L, x_1)$ is a convex function with respect to the parameter x_1. Let l_y^c be the current horizontal segment and assume without loss of generality that the decision algorithm moves right to a new segment $l_{y'}$; see Figure 3.8a. Consider a linear interpolation from l_y^c to $l_{y'}$. Let $l_{y''}$ be the segment at the midpoint of this linear interpolation. Since $D_{\pi[u,v]}(L, x_1)$ is a continuous function, for continuous functions, convex is the same as midpoint convex, this implies that we only need to show that $D_{\pi[u,v]}(L, x_1)$ is midpoint convex.

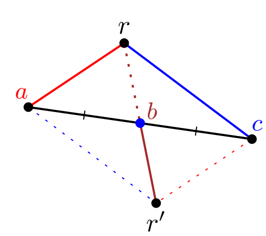
Consider the two mappings that minimise the Fréchet distance between $\pi\,[u, v]$ and the horizontal segments l_y^c and $l_{y'}$. Let r be any point on $\pi\,[u, v]$ and let a and c be the points where r is mapped to on l_y and $l_{y'}$. Construct a point b on $l_{y''}$ where r will be mapped to by linearly interpolating a and c. Performing this transformation for every point on $\pi[u, v]$, we obtain a valid mapping for $l_{y''}$, though not necessarily one of minimum Fréchet distance.

We bound the distance between r and b in terms of $\|ra\|$ and $\|rc\|$. Consider the parallelogram consisting of a, r, b, and a point r' that is distance $\|ra\|$ from c and distance $\|rc\|$ from a; see Figure 3.8b. Since b is the midpoint of ac, it is also the midpoint of rr' in this parallelogram. We can conclude that $\|rb\| \leq (\|ra\| + \|rc\|)/2$ in this mapping.

Since this property holds for any point r on $\pi[u, v]$ and the Fréchet distance is the minimum over all possible mappings, the Fréchet distance of $l_{y''}$ is upper bounded by the average of the Fréchet distances of l_y^c and $l_{y'}$. Therefore, the decision problem is convex.



(a) Point r is mapped to the three line segments.



(b) Upper bounding $\|rb\|$.

Figure 3.8: The decision algorithm is a convex function with respect to the left endpoint of the line segment.

Critical Values. An x-coordinate x_1 is a critical value of a decision algorithm if the decision algorithm could feasibly return that the left endpoint of l

has x-coordinate x_1.

For the C_1 class, these critical values are determined by up to three C_1 terms: the vertices themselves, the midpoint of any pair of vertices, and the center of the circle through the three (translated) points determining the Fréchet distance. Since each term in C_1 consists of at most n points, there are $\mathcal{O}(n^3)$ critical values in *Case 1*.

For the C_2 class, these critical values are the x-coordinates of the intersection points and minima of two hyperbolic functions, one for each element of each pair of two terms. Therefore, there are only a constant number of critical values for each two terms. Each term gives rise to at most $\mathcal{O}(n^2)$ hyperbolic functions (specifically, $B(\pi[u, v], l_y)$ can be of size $\Theta(n^2)$ in the worst case). Thus, there are at most $\mathcal{O}(n^4)$ critical values in *Case 2*.

Using similar arguments, it can be shown that there are at most $\mathcal{O}(n^4)$ critical values in *Case 3*, as they consist of at most two C_1 terms and at most one C_2 term.

Query Complexity. The decision algorithm runs in $T_s = \mathcal{O}(\log^{16} n)$ time since we use Theorem 2 to compute the optimal placement for a fixed left endpoint. The parallel form of the decision algorithm runs on one processor in $T_p = \mathcal{O}(\log^{16} n)$ time. Substituting these values in the running time of the parametric search of $\mathcal{O}(PT_p + T_pT_s \log P)$ leads to $\mathcal{O}(\log^{32} n)$ time.

Preprocessing and Space. Since we use the algorithm of Theorem 2 to the optimal placement of l^c for a given x-coordinate of its left endpoint, this requires $\mathcal{O}(n^2 \log^2 n)$ space and preprocessing time. □

3.5 Concluding Remarks

In this paper, we answer Translation Invariant Frechet distance queries between a horizontal query segment and a subtrajectory of a preprocessed trajectory. The most closely related result is that of De Berg *et al.* [18], which computes the normal Fréchet distance between a subtrajectory and a horizontal query segment. We extend this work in two way. Firstly, we consider all subtrajectories, not just vertex-to-vertex subtrajectories. Secondly, we compute the optimal translation for minimising the Fréchet distance, thus our approach allows us to compute both the normal Fréchet distance and the Translation Invariant Fréchet distance. All our queries can be answered in polylogarithmic time.

In terms of future work, one avenue would be to improve the query times. While our approach has polylogarithmic query time, the $\mathcal{O}(\log^{32} n)$ time needed for querying the optimal placement under translation is far from practical. Furthermore, our results use a $\mathcal{O}(n^2 \log^2 n)$ size data structure and reducing this would make the approach more appealing.

Other future work takes the form of generalising our queries further. In our most general form, we still work with a fixed length line segment with a fixed orientation. An interesting open problem is to see if we can also determine the optimal length of the line segment efficiently at query time. Allowing the line segment to have an arbitrary orientation seems a difficult problem to generalise our approach to, since the data structures we use assume that the line segment is horizontal. This can be extended to accommodate a constant number of orientations instead, but to extend this to truly arbitrary orientations, given at query time, will require significant modifications and novel ideas.

Bibliography

[1] Pankaj K. Agarwal and Jivr'i Matouvsek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993.

[2] Pankaj K. Agarwal and Micha Sharir. Efficient Algorithms for Geometric Optimization. *ACM Comput. Surv.*, 30(4):412–458, 1998.

[3] Miklós Ajtai, János Komlós, and Endre Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 1–9, 1983.

[4] Helmut Alt. The computational geometry of comparing shapes. In *Efficient Algorithms*, pages 235–248. Springer, 2009.

[5] Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(2):75–91, 1995.

[6] Helmut Alt, Christian Knauer, and Carola Wenk. Matching polygonal curves with respect to the Fréchet distance. In *STACS 2001, 18th Annual Symposium on Theoretical Aspects of Computer Science, Dresden, Germany, February 15-17, 2001, Proceedings*, pages 63–74, 2001.

[7] Elliot Anshelevich, Onkar Bhardwaj, and John Postl. Approximating optimal social choice under metric preferences. In *Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015*, pages 777–783, 2015.

[8] Duncan Black. On the rationale of group decision-making. *Journal of Political Economy*, 56(1):23–34, 1948.

[9] Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science*, pages 661–670, 2014.

[10] Karl Bringmann and Marvin Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *International Journal of Computational Geometry & Applications*, 27(1-2):85–120, 2017.

[11] Karl Bringmann, Marvin Künnemann, and André Nusse. Fréchet distance under translation: Conditional hardness and an algorithm via offline dynamic grid reachability. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2902–2921, 2019.

[12] Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017.

[13] Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, 1987.

[14] Richard Cole, Jeffrey S. Salowe, William L. Steiger, and Endre Szemerédi. An optimal-time algorithm for slope selection. *SIAM J. Comput.*, 18(4):792–810, 1989.

[15] Richard Cole, Micha Sharir, and Chee-Keng Yap. On k-hulls and related problems. *SIAM J. Comput.*, 16(1):61–77, 1987.

[16] Mark de Berg, Atlas F. Cook, and Joachim Gudmundsson. Fast Fréchet queries. *Computational Geometry*, 46(6):747–755, 2013.

[17] Mark de Berg, Joachim Gudmundsson, and Mehran Mehr. Faster algorithms for computing plurality points. In *32nd International Symposium on Computational Geometry, SoCG 2016*, pages 32:1–32:15, 2016.

[18] Mark De Berg, Ali D Mehrabi, and Tim Ophelders. Data structures for Fréchet queries in trajectory data. In *Proceedings of the 29th Canadian Conference on Computational Geometry*, 2017.

[19] Tamal K. Dey. Improved bounds on planar k-sets and k-levels. In *38th Annual Symposium on Foundations of Computer Science, FOCS 1997*, pages 156–161, 1997.

[20] Anthony Downs. An economic theory of political action in a democracy. *Journal of Political Economy*, 65(2):135–150, 1957.

[21] Anne Driemel and Sariel Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.

[22] James M Enelow and Melvin J Hinich. *The spatial theory of voting: An introduction.* CUP Archive, 1984.

[23] Chenglin Fan and Benjamin Raichel. Computing the Fréchet gap distance. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, pages 42:1–42:16, 2017.

[24] Scott L. Feld and Bernard Grofman. A theorem connecting Shapley-Owen power scores and the radius of the yolk in two dimensions. *Social Choice and Welfare*, 7(1):71–74, 1990.

[25] Scott L. Feld, Bernard Grofman, Richard Hartly, Marc Kilgour, and Nicholas Miller. The uncovered set in spatial voting games. *Theory and Decision*, 23(2):129–155, 1987.

[26] Scott L. Feld, Bernard Grofman, and Nicholas Miller. Centripetal forces in spatial voting: on the size of the yolk. *Public Choice*, 59(1):37–50, 1988.

[27] Scott L. Feld, Bernard Grofman, and Nicholas R. Miller. Limits on agenda control in spatial voting games. *Mathematical and Computer Modelling*, 12(4-5):405–416, 1989.

[28] Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 22(1):1–72, 1906.

[29] Joseph Godfrey. Computation of the Shapley-Owen power index in two dimensions. In *4th Annual workshop, University of Warwick*, pages 20–22, 2005.

[30] Ashish Goel, Anilesh Kollagunta Krishnaswamy, and Kamesh Munagala. Metric distortion of social choice rules: Lower bounds and fairness properties. In *Proceedings of the 2017 ACM Conference on Economics and Computation, EC 2017*, pages 287–304, 2017.

[31] Joachim Gudmundsson, Majid Mirzanezhad, Ali Mohades, and Carola Wenk. Fast Fréchet distance between curves with long edges. In *Proceedings of the 3rd International Workshop on Interactive and Spatial Computing*, pages 52–58, 2018.

[32] Joachim Gudmundsson and Michiel Smid. Fast algorithms for approximate Fréchet matching queries in geometric trees. *Computational Geometry*, 48(6):479–494, 2015.

[33] Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure-structure alignment with discrete Fréchet distance. *J. Bioinformatics and Computational Biology*, 6(1):51–64, 2008.

[34] Eamonn J. Keogh and Michael J. Pazzani. Scaling up dynamic time warping to massive datasets. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 1–11, 1999.

[35] David H. Koehler. The size of the yolk: computations for odd and even-numbered committees. *Social Choice and Welfare*, 7(3):231–245, 1990.

[36] Patrick Laube. *Computational Movement Analysis*. Springer Briefs in Computer Science. Springer, 2014.

[37] Richard D. McKelvey. Intransitivities in multidimensional voting models and some implications for agenda control. *Journal of Economic Theory*, 12(3):472 – 482, 1976.

[38] Richard D. McKelvey. Covering, dominance, and institution-free properties of social choice. *American Journal of Political Science*, pages 283–314, 1986.

[39] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. In *22nd Annual Symposium on Foundations of Computer Science (FOCS 1981)*, pages 399–408. IEEE, 1981.

[40] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

[41] Wouter Meulemans. *Similarity measures and algorithms for cartographic schematization*. PhD thesis, Technische Universiteit Eindhoven, 2014.

[42] Nicholas R. Miller. A new solution set for tournaments and majority voting: Further graph-theoretical approaches to the theory of voting. *American Journal of Political Science*, pages 68–96, 1980.

[43] Nicholas R. Miller, Bernard Grofman, and Scott L. Feld. The geometry of majority rule. *Journal of Theoretical Politics*, 1(4):379–406, 1989.

[44] Peter C. Ordeshook. The spatial analysis of elections and committees: Four decades of research. Technical report, California Institute of Technology, Division of the Humanities and Social Sciences, 1993.

[45] Charles R. Plott. A notion of equilibrium and its possibility under majority rule. *The American Economic Review*, pages 787–806, 1967.

[46] Keith T. Poole and Howard Rosenthal. The polarization of American politics. *The Journal of Politics*, 46(4):1061–1079, 1984.

[47] Keith T. Poole and Howard Rosenthal. Patterns of congressional voting. *American Journal of Political Science*, pages 228–278, 1991.

[48] Keith T. Poole and Howard Rosenthal. D-nominate after 10 years: A comparative update to congress: A political-economic history of roll-call voting. *Legislative Studies Quarterly*, pages 5–29, 2001.

[49] Franco P. Preparata. New parallel-sorting schemes. *IEEE Trans. Computers*, 27(7):669–673, 1978.

[50] Peter Ranacher and Katerina Tzavella. How to compare movement? A review of physical movement similarity measures in geographic information science and beyond. *Cartography and Geographic Information Science*, 41(3):286—307, 2014.

[51] Chotirat Ann Ratanamahatana and Eamonn Keogh. Three myths about dynamic time warping data mining. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 506–510, 2005.

[52] Norman Schofield. Equilibrium in the spatial 'valence' model of politics. *Journal of Theoretical Politics*, 16(4):447–481, 2004.

[53] Norman Schofield. *The spatial model of politics.* Routledge, 2007.

[54] Norman Schofield, Gary Miller, and Andrew Martin. Critical elections and political realignments in the USA: 1860–2000. *Political Studies*, 51(2):217–240, 2003.

[55] Micha Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete & Computational Geometry*, 18(2):125–134, 1997.

[56] Piotr Krzysztof Skowron and Edith Elkind. Social choice under metric preferences: Scoring rules and STV. In *Thirty-First AAAI Conference on Artificial Intelligence, AAAI 2017*, pages 706–712, 2017.

[57] E. Sriraghavendra, K. Karthik, and Chiranjib Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Proceedings of the 9th International Conference on Document Analysis and Recognition*, volume 1, pages 461–465, 2007.

[58] Richard E. Stone and Craig A. Tovey. Limiting median lines do not suffice to determine the yolk. *Social Choice and Welfare*, 9(1):33–35, 1992.

[59] Kevin Toohey and Matt Duckham. Trajectory similarity measures. *SIGSPATIAL Special*, 7(1):43–50, 2015.

[60] Géza Tóth. Point sets with many k-sets. In *16th Annual Symposium on Computational Geometry, SoCG 2000*, pages 37–42, 2000.

[61] Craig A. Tovey. A polynomial-time algorithm for computing the yolk in fixed dimension. *Mathematical Programming*, 57(1):259–277, 1992.

[62] Craig A. Tovey. Some foundations for empirical study in the Euclidean spatial model of social choice. In *Political economy: institutions, competition, and representation: Proceedings of the Seventh International Symposium in Economic Theory and Econometrics*, page 175. Cambridge Univ Pr, 1993.

[63] Craig A. Tovey. The almost surely shrinking yolk. *Mathematical Social Sciences*, 59(1):74–87, 2010.

[64] Craig A. Tovey. A finite exact algorithm for epsilon-core membership in two dimensions. *Mathematical Social Sciences*, 60(3):178–180, 2010.

[65] Craig A. Tovey. The Finagle point and the epsilon-core: a comment on Bräuninger's proof. *Journal of Theoretical Politics*, 23(1):135–139, 2011.

[66] Haozhou Wang, Han Su, Kai Zheng, Shazia Sadiq, and Xiaofang Zhou. An effectiveness study on trajectory similarity measures. In *Proceedings of the 24th Australasian Database Conference*, pages 13–22, 2013.

[67] Carola Wenk. *Shape matching in higher dimensions.* PhD thesis, Free University of Berlin, Dahlem, Germany, 2003.

[68] A. Wuffle, Scott L. Feld, Guillermo Owen, and Bernard Grofman. Finagle's law and the finagle point, a new solution concept for two-candidate competition in spatial voting games without a core. *American Journal of Political Science*, pages 348–375, 1989.

[69] Tim Wylie and Binhai Zhu. Protein chain pair simplification under the discrete Fréchet distance. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 10(6):1372–1383, 2013.