THE DISCRETE FRÉCHET DISTANCE WITH APPLICATIONS

by

Timothy Randall Wylie

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

July 2013

APPROVAL

of a dissertation submitted by

Timothy Randall Wylie

This dissertation has been read by each member of the dissertation committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to The Graduate School.

Dr. Binhai Zhu

Approved for the Department of Computer Science

Dr. John Paxton

Approved for The Graduate School

Dr. Ronald W. Larsen

STATEMENT OF PERMISSION TO USE

In presenting this dissertation in partial fulfillment of the requirements for a doctoral degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library. I further agree that copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for extensive copying or reproduction of this dissertation should be referred to ProQuest Information and Learning, 300 North Zeeb Road, Ann Arbor, Michigan 48106, to whom I have granted "the exclusive right to reproduce and distribute my dissertation in and from microform along with the non-exclusive right to reproduce and distribute my abstract in any format in whole or in part."

Timothy Randall Wylie

July 2013

DEDICATION

To Rachel, Maya, Casal, Elinor, and Eamon.

ACKNOWLEDGEMENTS

I was extremely fortunate to be taken as a student by Dr. Binhai Zhu. He has always assumed I was capable and pushed me to discover and learn independently. He ensured that I met deadlines, that I understood what needed to be done, and that I had funding. I am very grateful for his guidance and motivation, and his patience with my zealous ignorance.

I am thankful to Dr. Rafal Angryk for allowing me to share in his lab for a year, and for his encouragement and advice while therein. I am also grateful to Dr. Rocky Ross, Dr. Brendan Mumey, Dr. John Paxton, Dr. Qing Yang, Ms. Kathryn Hollenback, Ms. Shelly Shroyer, and Ms. Jeannette Radcliffe who have all helped me during my time at Montana State University and have all been eager to engage me in conversation regardless of the topic.

I would like to express my gratitude to my family who has always supported and encouraged me. Most importantly, I am forever indebted to my lovely wife, Rachel, who has far exceeded me in work and sacrifice for this degree. Finally, I owe my appreciation to my children. They have made hard days enjoyable, and good days unbelievable. I will never feel that I have failed while I am their father and they believe in me. They constantly remind me of the joy in wonder.

Funding Acknowledgment

This research was partially supported by NSF under grant DMS-0918034 and by NSF of China under project 60928006.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF ALGORITHMS

Algorithm Page

ABSTRACT

Modern computational geometry plays a critical role across a vast number of diverse research fields where theoretical results for provably efficient algorithms are necessary. Many of these problems are based on matching geometric objects or finding paths through given points with polygonal curves. This work focuses on the study and application of polygonal curves with respect to the discrete Fréchet distance.

We look at protein backbone structure alignment, comparison, and simplification. Previous work has largely focused on using the RMSD (Root Mean Square Deviation) distance measure. We build upon recent work demonstrating the benefits of the discrete Fréchet distance in this area, and present the first alignment algorithm based on the discrete Fréchet distance and compare our results with previous work.

To address visualization, the chain pair simplification problem (CPS-3F) was proposed in 2008 to simultaneously simplify two chains with respect to each other under the discrete Fréchet distance. The complexity of CPS-3F is unknown, so we present algorithms to address CPS-3F instances efficiently. We give a greedy backtracking heuristic and two factor-2 approximation algorithms for CPS-3F. We give empirical results for the heuristic and for one of the approximations, CPS-3F$^+$. Further, we provide dynamic programming solutions for many other CPS-3F properties. Chain pair simplification based on the Hausdorff distance (CPS-2H) is known to be **NP**-complete, and we prove the constrained version (CPS-2H$^+$) is also **NP**-complete.

We then investigate the discrete map matching and discrete set-chain matching problems and variations of them. The map matching problem is to find a path in a graph with a minimal Fréchet distance to a given polygonal line. The set-chain matching problem attempts to find another polygonal curve with nodes from a given point set. We prove the complexities of many of these problem variations when given a maximal number of vertices or points allowed, and when the paths are unique. Some of which are **NP**-complete and others we give a dynamic programming solution to.

Finally, many of the algorithms that we developed have also been implemented and released as a software library, named The Fréchet-based Protein Alignment & Comparison Toolkit (FPACT).

CHAPTER 1

INTRODUCTION

1.1 Motivation

Modern computational geometry plays a critical role across a vast number of diverse research fields. Theoretical results with efficient algorithms, whether heuristic, approximate, fixed-parameter tractable, etc., are necessary and applied in such disparate areas as protein structure alignment, Wi-Fi hotspot placement, pattern matching, computer vision, map routing and other GIS services, speech and handwriting processing, and a countless number of other applications.

Many of these problems are based on matching geometric objects and finding paths through designated points. The problems largely deal with, or can be abstracted to, polygonal curves, which we will focus on. The study of polygonal curves is not only imperative to many geometric applications, but some of these path problems are also fundamental, such as ordering, and are used to define complexity classes and completeness.

There are many measures that are commonly used with parametric curves. The Fréchet distance is often used when comparing two curves and can be described as a dissimilarity measure because the distance is a measure of how different the two curves are from each other [4]. When we examine polygonal curves, this calculation is much easier and can be done efficiently in $O(mn \log mn)$ time [5]. However, for many problems we are often only concerned with the nodes along the path and not the edges. For instance, our data may be sampled from a time series and thus we have ordering, but we may not be able to accurately infer information between the

sampled data. This led to the discrete Fréchet distance, which was first defined in the early 1990s [6].

Despite being well-known, the discrete Fréchet distance has not been studied or applied in many areas where other measures – such as the standard Fréchet distance, root mean square deviation, or the Hausdorff distance – are considered the standard benchmarks. Motivated by some biological, visualization, and map routing applications, we utilize the discrete Fréchet distance and show many positive results. Further, we analyze many unique aspects of the discrete Fréchet distance.

1.2 Overview

We begin with an overview of the research and many of the problems that are covered in the dissertation.

The discrete Fréchet distance is only a measure of the similarity of two polygonal curves from their current position in space, so we begin our synopsis with the alignment of the two curves first. The optimal alignment problem, as defined in [1], between two 3D chains under the discrete Fréchet distance takes $O(m^7 n^7 \log(m+n))$ time to solve [1]. Due to the high time complexity they proposed a heuristic method not dependent on the discrete Fréchet distance. In our first publication [2], we revisited the optimal alignment problem by proposing a possible PTAS heuristic algorithm in which all translations and rotations were based on the current discrete Fréchet distance of the two chains. We also showed that this was at worst a 2-approximation algorithm for the optimal alignment problem. The new algorithm provided better alignment results than the previous method for all empirical evaluations. This is reviewed in Chapter 3.

We also focused on another problem related to pairs of polygonal chains: simplification. Assuming two chains are optimally aligned, in what meaningful way can we simplify them with respect to each other? Can the chains be simplified such that their relationship maintains certain qualities of their similarity after simplification? Further, what are these qualities and how are they useful? To shed light on these questions, we utilized the Chain Pair Simplification (CPS) problem (Section 2.5) [7]. Specifically, CPS-3F – where the two chains and their comparison to each other are all simplified via the discrete Fréchet distance (we address other variations later).

The initial motivation for this problem was visualization of aligned protein backbones. Given that protein backbones can have as many as 500∼600 vertices (α-carbon atoms) in each chain, even with an optimal alignment, visualizing the similarity of two chains is difficult unless those chains are nearly identical. Our initial approach for CPS-3F was an efficient $O(n)$ greedy back-tracking algorithm [2]. This heuristic method was useful for efficiently handling pairs of extremely long chains. Using this heuristic with our alignment algorithm yielded positive results when empirically evaluated on protein backbone chains. However, the greedy nature of the method makes evaluating and controlling the simplification between the two chains difficult, even though the intra-chain simplifications are well-behaved. Chapter 4, Section 4.2 explains the algorithm and results in detail.

Our next approach was a dynamic programming algorithm which we showed was at worst a 2-approximation to an optimal CPS-3F simplification [3, 8]. We achieved this by defining the *moving cost* (Section 2.6) of the discrete Fréchet distance between two chains, which is a property controlled by weakly increasing integer sequences [9], but had never been defined or used as a measure on the discrete Fréchet distance. This measure is also unique to the discrete Fréchet distance, and thus is not a special case of the continuous Fréchet distance because it is infinite between continuous

curves. By minimizing the moving cost, we could exploit the inter and intra chain relationships and we greatly improved on all of our previous empirical results. The dynamic programming algorithm is not as efficient though, with a complexity between $O(mn)$ and $O(m^2n^2)$ depending on the input simplification parameters. The majority of Chapter 4 is devoted to this solution.

Further, chain pair simplification under the Hausdorff distance (CPS-2H) is **NP**-complete, and so we examined the complexity of CPS-2H under the constraint of the moving cost. We proved that under this new measure, the problem is still **NP**-complete [8], and this proof is given in Section 4.5.

Continuing in this vein of research, we have identified several other measurable properties of the discrete Fréchet distance which can also be evaluated by similar dynamic programming tactics. These have not been published and are discussed in Chapter 5. Here, we prove that one of these properties, the minimum sum of vertices for both chains, is also a factor-2 approximation for CPS-3F (Section 5.3.3). Further, we discuss the number of walks, one-sided optimal simplification, and the integral version. We also give an optimal solution for the minimum sum of the distances of pairs of vertices in the simplified chains. Without the simplification, this is equivalent to dynamic time warping (DTW) which is covered in detail in Section 2.7. The relationship between the discrete Fréchet distance and dynamic time warping is known [10], but little work has been done analyzing their similarities and differences, and no work has been done with dual simplification.

In evaluating these properties, the latter part of Chapter 5 focuses on alternative approaches to CPS-3F. Primarily with converting CPS-3F to a weighted directed acyclic graph, and then looking at several properties of the graph. Overall, this approach did not yield any new theoretical results, but may be beneficial in understanding CPS-3F and for eventually lowering the complexity of our 2-approximations.

To facilitate research using the discrete Fréchet distance we created a set of open-source libraries to run any of our algorithms based on the discrete Fréchet distance. The Fréchet-based Protein Alignment & Comparison Toolkit (FPACT) libraries were designed for easy access to the algorithms by being modular and format independent. The libraries are written and available in both C# and Python [11], and we give a brief synopsis of them in Chapter 8.

Another area of our investigation of the discrete Fréchet distance deals with the discrete set-chain matching and map matching problems (Chapters 6 and 7, respectively). These problems were originally defined and analyzed based on the continuous Fréchet distance [12, 13]. We not only examine them based on the discrete Fréchet distance, but also generalize the problem definitions with new variations.

One application of the map matching problem is recreating the most likely path of a vehicle on a road network given noisy GPS tracking data. Our work extends this analogy to assume the GPS data may also be intermittent. Suppose that in certain areas we have no connection with the GPS until some other point in time– we still have a polygonal curve, but we can not depend on all edges of the line to be accurate location data. Or similarly, situations exist where a person may check in periodically, but keeping a constant connection is too costly due to coverage or power constraints.

The discrete set-chain problem could be viewed as finding cellular towers to ensure coverage, given the route to travel and the maximum range of a tower. In the set-chain matching problem, there is a polygonal curve P, a set of points S, and an $\varepsilon > 0$ given. The problem is to find another polygonal curve Q such that the nodes of Q are points in S and the discrete Fréchet distance is $d_F(P, Q) \leq \varepsilon$. The map matching problem is similar, except instead of a set of points, we have a planar graph G and we must find a path through the graph that preserves the distance constraint. These are formally defined with our generalizations in Chapters 6 and 7.

We extend the problems in a few ways. The original works allow points/vertices to be used multiple times in Q, and they are only concerned whether such a path exists. We state the problems as minimization problems where we look at minimizing either the nodes of Q or the set of points/vertices from S/G used as nodes of Q. These still assume non-unique nodes in Q, so we also examine the problems given unique nodes, i.e., any point or vertex can only be used as a node in Q once. Note that when looking at unique nodes, minimizing $|Q|$ is equivalent to minimizing the set of points/vertices since they can only be used once. We provide optimal solutions via dynamic programming for the problems which are polynomial, and we prove that several of the problems are **NP**-complete.

In Chapter 9 we conclude with a summary of many of the main results.

CHAPTER 2

PRELIMINARIES

Here, we briefly overview the background material necessary for the work presented. References are also given for a more complete and thorough treatment of the concepts being used.

2.1 Polygonal Curves

We first overview the parametric definition of polygonal curves since our research is based on an investigation of this fundamental geometric structure. We look at the natural parameterization of polygonal curves [5, 14, 9, 13].

Let $\alpha : [0, p] \to \mathbb{R}^d$ be a polygonal curve in $\mathbb{R}^d$, which consists of p line segments $\bar{\alpha}_i := \alpha|_{[i,i+1]}$ for $i \in \{0, 1, \ldots, p-1\}$. Each line segment $\bar{\alpha}_i$ is affine and parameterized by its natural parameterization, i.e., $\alpha(i+\lambda) = (1-\lambda)\alpha(i)+\lambda\alpha(i+1)$ for all $\lambda \in [0, 1]$.

2.2 The Fréchet Distance

The Fréchet distance was first defined by Maurice Fréchet in 1906 as a measure of similarity between two parametric curves [4]. Subsequently, it has become a standard measure between parametric curves used in many areas. The Fréchet distance is typically explained as the relationship between a person and a dog connected by a leash walking along the two curves and trying to keep the leash as short as possible. The maximum length the leash reaches is the value of the Fréchet distance. This common analogy has led to the term "dog-walking distance" sometimes being used.

We first define parameterized continuous curves and then formally give the standard definition for the Fréchet distance [5, 14].

Definition 1. A continuous parameterized curve $A \in \mathbb{R}^d$ can be represented by a continuous mapping $f : [a, b] \to \mathbb{R}^d$ such that $a, b \in \mathbb{R}$ and $a < b$.

Definition 2. A monotone reparameterization α is a continuous non-decreasing function $\alpha : [0, 1] \to [0, 1]$ such that $\alpha(0) = 0$ and $\alpha(1) = 1$.

Definition 3. Given two curves, A, B in a metric space, the **Fréchet distance**, $d_{\mathcal{F}}(A, B)$ is defined as

$$d_{\mathcal{F}}(A, B) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \{d(A(\alpha(t)), B(\beta(t)))\}$$

where α, β range over all monotone reparameterizations and $d(\cdot, \cdot)$ represents the Euclidean distance, and inf is the infimum.

In the early 1990s, the Fréchet distance was applied to polygonal curves by Alt and Godau [15, 5]. With the restriction of polygonal curves, they proved the Fréchet distance can be found between two curves A, B efficiently with a time complexity of $O(mn \log mn)$ where $m = |A|, n = |B|$. Godau also showed that the decision version of the problem could be solved in $O(mn)$ time [16].

2.3 The Discrete Fréchet Distance

In 1994 Eiter and Mannila defined the discrete Fréchet distance as an approximation of the Fréchet distance to be used between two polygonal chains using only the nodes along the chains for the measurements [6]. They also referred to this discrete form as the coupling distance which is used synonymously. Furthermore, they proved

the discrete version can be computed in $O(mn)$ time where m, n are the number of vertices in each polygonal chain. A rigorous look at the definition of the discrete Fréchet distance was also done by Mosig et al. in 2005 [9].

We first give the formal definition, and then an equivalent alternate definition that is beneficial for our research. We use $d(a, b)$ to represent the Euclidean distance between two 3D points a and b, but it can be replaced with another distance measure, depending on applications.

Definition 4. The discrete Fréchet distance d_F between two polygonal curves $f : [0, m] \to \mathbb{R}^k$ and $g : [0, n] \to \mathbb{R}^k$ is defined as:

$$d_F(f, g) = \min_{\sigma:[1:m+n]\to[0:m],\beta:[1:m+n]\to[0:n]} \max_{s\in[1:m+n]} \left\{ d\Big(f(\sigma(s)),\, g(\beta(s))\Big) \right\}$$

where σ and β range over all discrete non-decreasing onto mappings of the form $\sigma : [1 : m + n] \to [0 : m], \beta : [1 : m + n] \to [0 : n]$.

For our alternate definition, we use the graph-theoretic term "paths" instead of the geometric term "polygonal chains" because our definition makes no assumption that the underlying space of points is geometric. Given two paths, we define their discrete Fréchet distance as follows.

Definition 5. Given a path $P = \langle p_1, \ldots, p_n \rangle$ of n vertices, a **t-walk** along P is a partitioning of P along the path into t disjoint non-empty subpaths $\{P_i\}_{i=1..t}$ such that $P_i = \langle p_{n_{i-1}+1}, \ldots, p_{n_i} \rangle$ and $0 = n_0 < n_1 < \cdots < n_t = n$.

Definition 6. Given two paths $A = \langle a_1, \ldots, a_m \rangle$ and $B = \langle b_1, \ldots, b_n \rangle$, a **paired walk** along A and B is a t-walk $\{A_i\}_{i=1..t}$ along A and a t-walk $\{B_i\}_{i=1..t}$ along B for some t, such that, for $1 \leq i \leq t$, either $|A_i| = 1$ or $|B_i| = 1$ (that is, either A_i or B_i contains exactly one vertex).

Definition 7. The **cost** of a paired walk $W = \{(A_i, B_i)\}$ along two paths A and B is

$$d_F^W(A, B) = \max_i \max_{(a,b)\in A_i \times B_i} d(a, b).$$

Definition 8. The **discrete Fréchet distance** between two paths A and B is

$$d_F(A, B) = \min_W d_F^W(A, B).$$

A paired walk that achieves the discrete Fréchet distance between two paths A and B is called a **Fréchet alignment** of A and B.

If we revisit the dog-walking analogy, we consider the scenario in which a person walks along A and a dog along B. Intuitively, the definition of the paired walk is based on three cases:

1. $|B_i| > |A_i| = 1$: the person stays and the dog hops forward;
2. $|A_i| > |B_i| = 1$: the person hops forward and the dog stays;
3. $|A_i| = |B_i| = 1$: both the person and the dog hop forward.

The following figure shows the relationship between the discrete and continuous Fréchet distances. In Figure 2.1(a), we have the two chains $\langle a_1, a_2, a_3\rangle$ and $\langle b_1, b_2\rangle$, the continuous Fréchet distance between the two is the distance from a_2 to segment $\overline{b_1 b_2}$, i.e., $d(a_2, o)$. The discrete Fréchet distance is $d(a_2, b_2)$. The discrete Fréchet distance could be quite larger than the continuous distance. On the other hand, with enough sample points on the two chains, the resulting discrete Fréchet distance, i.e., $d(a_2, b)$ in Figure 2.1(b), closely approximates $d(a_2, o)$.

With enough evenly sampled nodes the discrete Fréchet distance can closely approximate the continuous version. The relationship between the discrete Fréchet

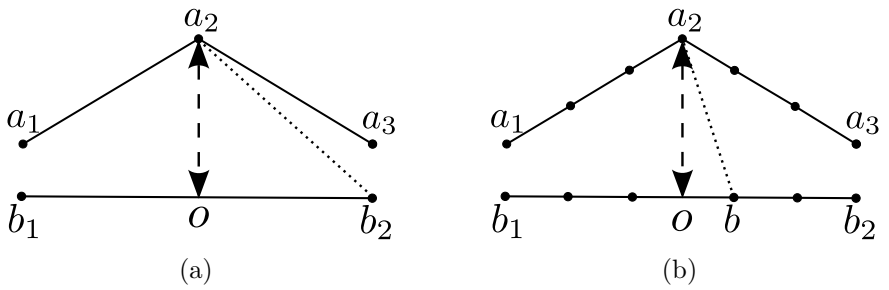


Figure 2.1: The relationship between the discrete and continuous Fréchet distance where o is the continuous and the dotted line between nodes is the discrete. (a) shows a case where the chains have fewer nodes and a larger discrete Fréchet distance, while (b) is the same path with more nodes, and thus provides a better approximation of the Fréchet distance.

distance (d_F) and the continuous Fréchet distance ($d_\mathcal{F}$) is known to be bounded by the following equation.

$$d_\mathcal{F}(A, B) \leq d_F(A, B) \leq d_\mathcal{F}(A, B) + \max\{l_a, l_b\}$$

where A and B are polygonal curves, and l_a, l_b are the longest edges on A, B, respectively [14, 6].

Eiter and Mannila showed that with a standard dynamic programming approach, it is not hard to obtain the following theorem for the discrete Fréchet distance.

Theorem 1. *[6] The discrete Fréchet distance between two paths with m and n vertices respectively can be computed in $O(mn)$ time.*

In two dimensions, it was recently shown that subquadratic time is possible, but the difference is marginal [17]. The new algorithm still requires $O(\frac{mn \log\log n}{\log n})$ time.

2.4 The Hausdorff Distance

The Hausdorff distance was first defined by Felix Hausdorff in 1914 [18]. Since its introduction, the Hausdorff distance has become one of the most widely used similarity measures across many disciplines. Definition 9 is taken from [19].

Definition 9. Let X and Y be two non-empty subsets of a metric space (M, d) where M is the space and d the distance measure. We define their Hausdorff distance $d_H(X, Y)$ by

$$d_H(X, Y) = \max\{\sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y)\},$$

where sup represents the supremum and inf the infimum.

In this work we largely focus on the Fréchet distance, but understanding the difference between the Hausdorff distance and the Fréchet distance is important. Figure 2.2 shows a classic example used to demonstrate this difference. There are two polygonal curves which intersect repeatedly. Due to this crossing the Hausdorff distance is less than or equal to δ, however, because the curves zigzag in different directions the Fréchet distance is greater than 2δ.

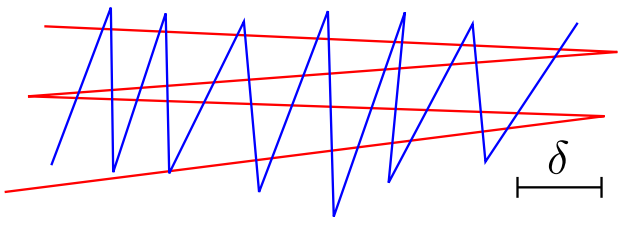


Figure 2.2: Two polygonal curves that highlight the difference between the Fréchet distance and the Hausdorff distance. Here $d_H \leq \delta$, but $d_F > 2\delta$.

This example shows how the Fréchet distance is more dependent on the actual flow of the curves themselves. The actual figure is based on the example given in [15] by Alt et al.

2.5 Chain Pair Simplification

In 2008, the chain pair simplification problem in three dimensions under the discrete Fréchet distance was defined in order to allow better visualization of two polygonal chains [7]. The problem not only allows better visualization of the two chains in a simplified form, but it also keeps and exploits the characteristic similarities that exist between the chains. Although the problem does not necessarily need to be limited to only 3D space, we state the original decision problem as it was defined relating to protein backbone chains.

Definition 10 The Chain Pair Simplification (CPS) Problem.
Instance: Given a pair of 3D chains A and B, with lengths $O(m), O(n)$ respectively, an integer $K > 0$, and three real numbers $\delta_1, \delta_2, \delta_3 > 0$.
Problem: Does there exist a pair of chains A', B', each of at most K vertices, such that the vertices of A', B' are from A, B respectively, and $d_1(A, A') \leq \delta_1, d_2(B, B') \leq \delta_2, d_F(A', B') \leq \delta_3$?

When $d_1 = d_2 = d_F$, the problem is called CPS-3F since all three distance measures are the discrete Fréchet distance. When $d_1 = d_2 = d_H$ (the Hausdorff distance), the problem is called CPS-2H since two of the distances are Hausdorff. CPS-2H was proven to be NP-complete [7], but the complexity of CPS-3F is unknown.

2.6 The Moving Cost

Here, we use the theory of weakly increasing integers [9] with simultaneous simplification to define what we call the moving cost of the alignment between two chains.

Definition 11. The **moving cost** of a paired walk $W = \{(A_i, B_i)\}$ is

$$m_c^W(A_i, B_i) = \max\{|A_i|, |B_i|\} \tag{2.1}$$

The moving cost of a paired walk W between A and B is

$$m_c^W(A, B) = \sum_{i=1}^{t} m_c^W(A_i, B_i). \tag{2.2}$$

The moving cost for A and B is the sum of the number of "hops" the man or dog make along the two chains. However, when they both move at once, this only counts as a single move. In other words, it is the number of pairs of points, or matched points, between the chains used in calculating the discrete Fréchet distance.

In Figure 2.3 we show a very simple example of two chains that can be simplified in two possible ways. In Figure 2.3(a) the moving cost is six and the number of nodes for each chain is four, and in Figure 2.3(b) the moving cost is still six, yet the number of nodes is now five in each chain.

We can prove some nice properties of the moving cost, such as the complexity being polynomial and its ability to approximate the number of vertices. As we make use of later, $\max(|A|, |B|) \leq m_c^W(A, B) \leq |A| + |B| - t$ for a paired t-walk W along A and B. This is because in any paired walk (A_i, B_i), the moving cost is at least $\max\{|A_i|, |B_i|\}$ and at most $|A_i| + |B_i| - 1$. This is the motivation for our variant of CPS-3F and CPS-2H.

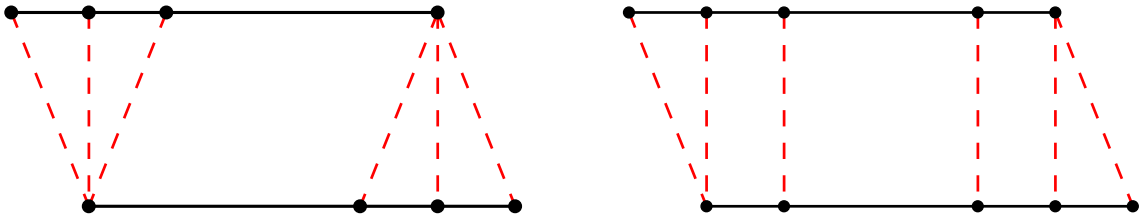


Figure 2.3: The difference between the number of nodes and the moving cost. Suppose that both (a) and (b) are valid simplifications of two chains. They have the same moving cost, yet (a) only has four nodes in each of the simplified chains, but in (b) both chains have five nodes.

Definition 12 The Constrained Chain Pair Simplification (CPS$^+$) Problem.

Instance: Given a pair of 3D chains A and B, with lengths $O(m), O(n)$ respectively, an integer $K' > 0$, and $\delta_1, \delta_2, \delta_3 \in \mathbb{R}^+$.

Problem: Does there exist a pair of chains A', B' where the vertices are from A, B, respectively, such that for some paired walk W between A', B', $m_c^W(A', B') \leq K'$, and $d_1(A, A') \leq \delta_1, d_2(B, B') \leq \delta_2, d_F(A', B') \leq \delta_3$?

When $d_1 = d_2 = d_F$, we call the problem CPS-3F$^+$, and when $d_1 = d_2 = d_H$, the problem is denoted CPS-2H$^+$.

2.7 Dynamic Time Warping

Another parametric distance measure is Dynamic Time Warping (DTW), and it was first introduced by Vintsyuk in 1968 in the field of speech processing [20]. The last 15 years it has been used largely in the data mining community as a method for comparing, indexing, and learning with temporal and spatial data.

Dynamic time warping has been developed extensively in many areas for specific applications. The idea is related to the discrete Fréchet distance, but differs in finding the minimum sum between points as opposed to the minimum maximum distance between any two pairs. We state the definition in a similar manner to the discrete Fréchet distance. Our formulation of the definitions is from Muller [21], which is recommended for a more thorough explanation of DTW and the surrounding research.

Definition 13. Given two sequences $A = \langle a_1, \ldots, a_m \rangle$ and $B = \langle b_1, \ldots, b_n \rangle$ in a feature space $\mathcal{F}$ and a cost function based on the distance function d. An (M, N)-**warping path** is a continuous monotone reparameterized sequence $p = \langle p_1, \ldots, p_l \rangle$ of A, B such that $p_1 = (a_1, b_1)$ and $p_l = (a_m, b_n)$.

Definition 14. The **total cost** of a warping path p between A and B is defined as

$$c_p(A, B) = \sum_{l=1}^{L} d(a_{m_l}, b_{n_l}).$$

Definition 15. Now the **optimal warping path** is a warping path having the minimal total cost among all feasible warping paths. Thus, **dynamic time warping** is defined as

$$DTW(A, B) = \min c_p(A, B),$$

for all feasible warping paths p.

The standard formal definition is based on the parameterization of the two sequences.

Definition 16. The dynamic time warping distance (DTW) between two polygonal curves $f : [0, m] \to \mathbb{R}^k$ and $g : [0, n] \to \mathbb{R}^k$ is defined as:

$$DTW(f, g) = \min_{\sigma:[0:Q]\to[0:m],\ \beta:[0:Q]\to[0:n]} \|f \circ \sigma,\ g \circ \beta\|_2$$

where σ and β range over all discrete non-decreasing onto mappings of the form $\sigma : [0 : Q] \to [0 : m]$, $\beta : [0 : Q] \to [0 : n]$ $\forall Q \in [\max(m, n), m + n]$.

To see the relationship between dynamic time warping and the discrete Fréchet distance, we can slightly modify the definition of the discrete Fréchet distance (Definition 4). Note that we can remove the max function if we are in an L_∞ normal space [22]. Then the definition is as follows:

Definition 17. The discrete Fréchet distance (d_F) between two polygonal curves $f : [0, m] \to \mathbb{R}^k$ and $g : [0, n] \to \mathbb{R}^k$ in the L_∞-norm is defined as:

$$d_F(f, g) = \min_{\sigma:[1:m+n]\to[0:m],\ \beta:[1:m+n]\to[0:n]} \|f \circ \sigma,\ g \circ \beta\|_\infty$$

where σ and β range over all discrete non-decreasing onto mappings of the form $\sigma : [1 : m + n] \to [0 : m]$, $\beta : [1 : m + n] \to [0 : n]$.

In a sense, we are only changing the L-norm in which we find the parametric distance. Both have advantages and disadvantages when comparing two polygonal curves. The combination of the two provides a much greater insight into how similar the two curves are.

CHAPTER 3

ALIGNMENT OF POLYGONAL CURVES

In this chapter we cover the research that we have done regarding protein backbone alignment. We overview the applications, related work, our motivation, the algorithm, and then provide our empirical results. These highlight the algorithm contributions and the benefits of our alignment.

3.1 Application and Related Work

The alignment and comparison of polygonal chains have been well studied in several fields including computer vision, bioinformatics, computational geometry, and parametric curve approximations [23, 24, 25]. Within structural biology, polygonal chain similarity is one of the central problems of protein research. In general, it is believed that a protein's structure implies its function, and thus to compare the functionality of proteins their structures must be compared [26]. This is known to be true for certain situations, especially with homologous traits between proteins, and in general the empirical evidence between proteins is in agreement [27, 26]. The structure is defined by the α-carbon atoms of the residues (amino acids) along the backbone of each chain. These atoms represent the vertices that constitute our 3D polygonal chains.

Since the structure of the protein is intrinsically related to its function, there have been many software systems designed for protein structure alignment and comparison in the last couple of decades. A few of the more well-known systems are MAMMOTH [28], SCOP [29], DALI [30, 31], CATH [32], CE [33], ProteinDBS [34], SSAP [35],

and 3D-BLAST [36]. None of these systems use the discrete Fréchet distance, and the majority of the work previously done on protein global structure alignment and protein local structure alignment uses the RMSD (root mean square deviation) distance measure. Given two m-vectors $V_1 = \langle u_1, u_2, ..., u_m \rangle$ and $V_2 = \langle v_1, v_2, ..., v_m \rangle$, RMSD is defined as:

$$\text{RMSD}(V_1, V_2) = \sqrt{\frac{\sum_i (u_i - v_i)^2}{m}}.$$

This gives an average pair-wise distance along the two vectors which provides some insight into the similarity of the two chains, but the reliance on m shows one of the major drawbacks of using RMSD. The comparison hinges on the necessity that the two vectors be the same length and that the vertices at a given index in each chain be pairwise similar. If we modified the chains at all we could receive very different RMSD values. Suppose we are given two chains C_1, C_2 with m vertices, and we then add some vertices on C_1 and C_2 by alternatively duplicating/repeating some different vertices in C_1 and C_2 to obtain C_1', C_2', then RMSD(C_1', C_2') could be dramatically different from RMSD(C_1, C_2), even though geometrically C_1' and C_2' are just as close as C_1 and C_2 are. This suggests that a measure independent of the number of vertices or a pair-wise alignment would be a better indication on the similarity of the two chains.

To achieve a more accurate measure of similarity between two protein structures, Jiang et al. proposed using the discrete Fréchet distance for the protein backbone comparison [1]. The two main problems they addressed were the alignment of the two chains, and then the comparison itself. They showed that the optimal alignment problem, as defined in [1], between two 3D chains under the discrete Fréchet distance takes $O(n^7 m^7 \log(n + m))$ time to solve [1]. Due to the high time complexity they

proposed a heuristic method not dependent on the discrete Fréchet distance. We compare the results with our algorithm and show an improvement in alignment.

3.2 Alignment Under the Discrete Fréchet Distance

The optimal (global structure-structure) alignment problem is formally defined as follows.

Definition 18. Given two 3D polygonal chains A and B, a transformation class T, and a distance measure $d(-)$, find a transformation $\tau \in T$ such that $dist(A, \tau(B))$ is minimized.

Of course, in our case T contains both rotation and translation, and $d = d_F$.

Let $A = \langle a_1, a_2, ..., a_m \rangle$ and $B = \langle b_1, b_2, ..., b_n \rangle$. It was shown that the optimal alignment problem under the discrete Fréchet distance can be solved in $O(n^7 m^7 \log(n+m))$ time [1]. This is impractical in use, so Jiang et al. presented a heuristic method which focuses on first aligning the center a of A and the center b of B. (Given a 3D chain C of n vertices, the coordinates of each vertex c_i of C is really a vector $\vec{c_i}$, the center c corresponds to $\vec{c} = \frac{\sum_i \vec{c_i}}{n}$.) Then a rotation is performed such that $\triangle a_1 a a_m$ and $\triangle b_1 b b_n$ are on the same plane. Finally, some local improvements are performed until the discrete Fréchet distance cannot be further improved. While this algorithm is still slower compared to some of the known software (like ProteinDBS [34]), it can improve the accuracy in many situations [1].

3.3 ALIGN Algorithm

We use a slightly different idea here. We can prove that if we first move B such that b_1 is located exactly at a_1 and subsequently obtain an optimal solution, then this solution is a factor-2 approximation for the optimal alignment problem (when a_1 does not necessarily collide with b_1). Of course, a factor-2 approximation may not be accurate enough for many biological applications. Therefore, while colliding b_1 at a_1 is our starting point, our algorithm goes beyond that. Our complete (heuristic) algorithm is as follows.

While we are unable to prove that this algorithm is a polynomial time approximation scheme (PTAS) for the optimal alignment problem, we believe that for practical data it is almost a PTAS. Now, we give some evidence that, when translating B such that b_1 collides with a_1 and obtaining subsequently an optimal solution (with b_1 sticking at a_1), in fact gives us a factor-2 approximation for the optimal alignment problem.

Lemma 1. Given two 3D polygonal chains A, B of length m, n respectively such that the optimal $d_F(A, B) = \epsilon$, an optimal transformation τ aligning A and $\tau(B)$ such that $d(a_1, \tau(b_1)) = 0$ gives a 2-approximation for the optimal alignment problem.

Proof. Let $a_i \in A$, $b_j \in B$ be the two vertices defining the optimal discrete Fréchet distance $d_F^*(A, B) = \epsilon$ for the optimal alignment of A and B. Then, by definition $d_F^*(A, B) = d(a_i, b_j)$; moreover, $d(a_1, b_1) \leq d_F^*(A, B) = \epsilon$. Now run a translation τ' for this optimal alignment such that $\tau'(b_1)$ collides at a_1. Then obviously $d_F(A, \tau'(B)) \leq \epsilon + d(a_1, b_1) \leq 2\epsilon$. As τ is an optimal transformation with the con-

Algorithm 3.1 ALIGN → Align two polygonal curves with rotation and translation based on the discrete Fréchet distance.

Algorithm ALIGN(A, B):
Input: Two polygonal chains $A = \langle a_1, ..., a_m \rangle$ and $B = \langle b_1, ..., b_n \rangle$.

1. Translate B so that $d(b_1, a_1) = 0$.
2. Let β be the midpoint of $\langle a_m, b_n \rangle$. Rotate B around the axis line (a_1, β) so that $d(a_m, b_n)$ is minimized. Let $a_i \in A$ and $b_j \in B$ be the two vertices such that $d(a_i, b_j) = d_F(A, B)$.
3. Initialize $O^*(A, B) \longleftarrow d_F(A, B)$.
4. Loop until no improvement of $O^*(A, B)$ is made.
   (a) Rotate until no improvement of $O^*(A, B)$ is made.
      i. Let γ be the midpoint between a_1, b_1. Let μ be the midpoint between a_i, b_j.
      ii. Rotate B around the axis line (γ, μ) by θ such that $-180 \leq \theta \leq 180$ and $|\theta|$ is the largest angle which results in $d_F(A, B) < O^*(A, B)$.
      iii. Update $O^*(A, B) \longleftarrow d_F(A, B)$ and update a_i, b_j accordingly.
   (b) Translate until no improvement of $O^*(A, B)$ is made.
      i. Translate B along the vector $\overrightarrow{b_j a_i}$ by δ such that δ is the largest value which results in $d_F(A, B) < O^*(A, B)$.
      ii. Update $O^*(A, B) \longleftarrow d_F(A, B)$ and update a_i, b_j accordingly.
5. Return $A, B, O^*(A, B)$.

straint that $d(a_1, \tau(b_1)) = 0$,

$$d_F(A, \tau(B)) \leq d_F(A, \tau'(B)) \leq 2\epsilon.$$

□

In our algorithm, while initially we force b_1 to collide with a_1 in Step 1, the rotations and translations will generally separate them from each other. More precisely, unless $d_F(A, B) = 0$, they will be separated during the first translation.

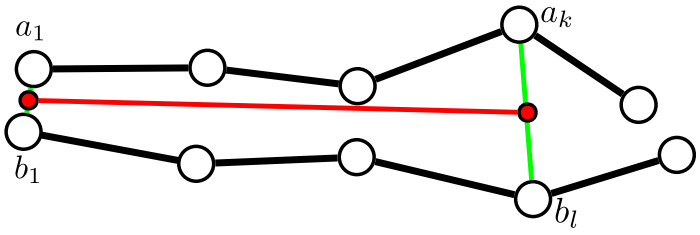


Figure 3.1: Example polygonal curves being aligned via rotation where the discrete Fréchet distance is defined by $d(a_k, b_l)$.

Figure 3.1 shows an example alignment during the rotation step. The midpoint between (a_k, b_l) has been found and the vector from that midpoint to the midpoint between (a_1, b_1) has been drawn. This is the rotation axis that will hopefully allow b_l to move closer to a_k and decrease the overall discrete Fréchet distance between A and B.

3.4 Empirical Results

In [1], rigorous studies are performed regarding comparing protein backbone 107j.a with the other seven chains from the Protein Database (PDB): 1hfj.c, 1qd1.b, 1toh, 4eca.c, 1d9q.d, 4eca.b, 4eca.d. These seven chains were reported to be similar to 107.j by the ProteinDBS software (which takes a few seconds searching the whole PDB, which contained over 30,000 protein backbones at that time). Using the discrete Fréchet distance as distance measure, while taking much longer (close to one minute for each pair), the heuristic algorithm in [1] reported that 3 of the 7 chains are in fact not really similar to 107.j. ProteinDBS subsequently updated their webpage for this. Here, in Table 3.1, we simply compare our ALIGN algorithm with that of [1]. We are concerned primarily with accuracy. All distances are measured in Ångströms.

Protein Chain (B)	RMSD [1]	$d_F(A, B)$ [1]	$d_F(A, B)$ [2]
1hfj.c	0.27	1.01	0.95
1qd1.b	2.81	22.90	22.65
1toh	2.91	35.09	22.06
4eca.c	1.10	6.01	5.55
1d9q.d	2.88	22.18	20.87
4eca.b	1.09	5.76	5.64
4eca.d	1.45	5.92	5.71

Table 3.1: Alignment with 107j.a (Chain A) where all eight chains have 325 vertices, and the original work is [1] and our new algorithm, ALIGN, is [2].

3.5 Possible Improvement

Although we will not spend a lot of space on possible improvements, it is beneficial to discuss promising roads that may be interesting from an efficiency and an algorithmic perspective.

Our algorithms make no assumptions about the input polygonal curves. However, for our application we have a lot of additional information at our disposal. Often additional knowledge to restrict the problem can be extremely beneficial. For proteins, we know that the α-carbon atoms of the backbone are evenly spaced. This allows us to do some simple sampling with bounds on the discrete Fréchet distance compared to the full chain.

If we decide to sample every k^{th} node, we still know approximately how good the alignment is, and we have a bound on how large the discrete Fréchet distance may be. Figure 3.2 shows a simple example of this where $k = 2$, and we have guaranteed that every node sampled is within δ of the other chains sampled nodes. If we let the fixed distance of all edges be x, then $d_F(A, B) \leq k \cdot x + \delta$, and the number of vertices

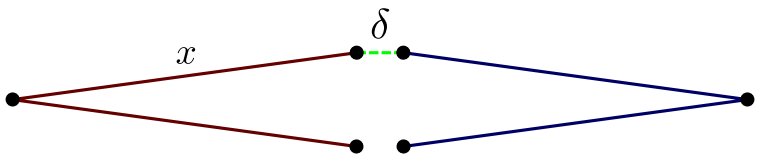


Figure 3.2: Segments of two chains with a fixed distance between nodes, and every other vertex aligned within δ of the other chain.

in a chain is $|A'| = |B'| \leq \left\lceil \frac{n}{s+1} \right\rceil$. This assumes that both polygonal curves have the same initial length, n, but this can be modified to be more flexible.

A dynamic version of this could be employed to iteratively align the curves from a low sampled to highly sampled solution while maintaining the continually improving bound on the discrete Fréchet distance.

Another area which might improve the alignment is to use dynamic time warping in conjunction with the discrete Fréchet distance. As noted in Section 2.7, the discrete Fréchet distance is intimately related to dynamic time warping. If we consider them in terms of an L-norm, DTW is in L_2 (Euclidean) and the discrete Fréchet distance is the same parameterization in L_∞. In fact, our alignment algorithm would change very little if based on DTW. The farthest pair between the two chains that is the $d_F(A, B)$ will often be the same pair as the farthest two based on DTW(A, B). This should allow the two chains to be better aligned as a whole. In a sense, dynamic time warping is an intermediate measure between the discrete Fréchet distance and RMSD. Thus, it may allow for a more accurate alignment and more dependable measure because outliers will affect it less and it still considers every vertex in each curve.

CHAPTER 4

CHAIN PAIR SIMPLIFICATION

Chain pair simplification is a significant portion of our discussion on the benefits of the discrete Fréchet distance. In this chapter we will overview our work with CPS-3F related to protein backbone simplification. After discussing the application in more detail, we detail a heuristic and an approximation algorithm for CPS-3F and compare them using the alignments discussed in the previous chapter.

4.1 Application

Given that protein backbones can have as many as 500∼600 vertices (α-carbon atoms) in each chain, even with an optimal alignment, visualizing the similarity of two chains is difficult unless those chains are nearly identical. To address this issue the chain pair simplification (CPS-3F) problem was proposed by Bereg et al. in 2008 [7]. They were unable to show whether CPS-3F is **NP**-complete, but they proved that the Hausdorff version of the problem (CPS-2H), which simplified the chains via the Hausdorff distance, is **NP**-complete [7]. This led them to postulate that under the discrete Fréchet distance the problem was likely to be **NP**-complete. In our previous work [2] we used a heuristic $O(n)$ time algorithm to simplify pairs of chains. Here, we prove that a variation of CPS-3F, denoted as CPS-3F$^+$, is polynomially solvable, and that it is a factor-2 approximation for CPS-3F. This restricted version is also beneficial because as we simplify the proteins, we usually want to visually compare the two backbone chains without distorting their lengths. However, we also look at cases where uneven simplification may be beneficial for visualization. Further, we

prove that the constrained version on CPS-2H, similarly denoted as CPS-2H$^+$, is **NP**-complete.

4.2 CPS-3F Heuristic

Here we try to solve the CPS-3F problem with a practical solution. It is known that the greedy method does not always work even for simplifying a single chain under the discrete Fréchet distance, with some counterexample presented in [7]. Here, we use a greedy backtracking method. Our ideas are as follows: (1) While greedy does not always work, for protein backbones we have the implicit condition that for all possible i $d(a_i, a_{i+1}) \approx 3.7$ to 3.8 (Ångströms), i.e., the neighboring α-carbon atoms in a protein backbone have an almost uniform length. With this condition a lot of counter examples do not hold anymore. (2) To mend possible holes of the algorithm, when we are stuck at a certain point (using the greedy method), we backtrack some (constant) number of steps and retry the greedy method again. While this algorithm does not generally lead to an optimal solution, it works well for practical protein data, and we present some of those results in Section 4.4.

We first show a simple lemma which helps us to determine whether the input could lead to an infeasible solution. Certainly, this lemma also implies that having an almost optimal alignment of A and B, resulting in minimizing $d_F(A, B)$, is crucial for the success of the simplification algorithm.

Lemma 2. Given two 3D polygonal chains A and B, if a solution (A', B') for CPS-3F is found with $d_F(A, A') \leq \delta_1$, $d_F(B, B') \leq \delta_2$ and $d_F(A', B') \leq \delta_3$, then $d_F(A, B) \leq \delta_1 + \delta_2 + \delta_3$.

4.2.1 SIMPLIFY Algorithm

Let $\mathcal{B}(b, \delta)$ be a ball centered at point b with radius δ. Our heuristic algorithm for CPS-3F assumes that A and B are almost optimally aligned. Step 1 of the algorithm calls the ALIGN algorithm covered in Chapter 3 if the two chains have not been aligned already.

The algorithm returns two simplified chains where $|A'| = |B'| = K$ whether they could be simplified more or not. Thus, it is possible for consecutive nodes of A', B' to be equal, e.g. $a_i = a_j = a_r$ where $a_i, a_j \in A', a_r \in A$. This duplication can easily be taken out if the desire is that the chains be less than or equal to K rather than only equal. Usually this difference in the simplification is insignificant, but it can be beneficial occasionally.

4.2.2 Empirical Results

The results for the heuristic algorithm are in Tables 4.2 and 4.3. The setup and values are discussed in detail in Section 4.4, so they can be easily compared with the results from our approximation algorithm.

4.3 CPS-3F$^+$ is a 2-approximation

The greedy nature of our heuristic method makes evaluating and controlling the simplification between the two chains difficult and far from optimal. To improve these results, we define a new metric on the discrete Fréchet distance called the *moving cost*. Using the moving cost, we define a dynammic programming algorithm that minimizes the new measure, and we prove that it is a 2-approximation algorithm for the optimal CPS-3F solution.

Algorithm 4.2 SIMPLIFY $\rightarrow$ Simplify the two curves for CPS-3F with a greedy backtracking method.

Algorithm SIMPLIFY$(A, B, K, \delta_1, \delta_2, \delta_3)$:
Input: Two polygonal chains $A = \langle a_1, ..., a_m \rangle$ and $B = \langle b_1, ..., b_n \rangle$, a positive integer K, and three positive constants $\delta_1, \delta_2, \delta_3$.
Output: Two simplified chains $A' = \langle a'_1, ..., a'_K \rangle$ and $B' = \langle b'_1, ..., b'_K \rangle$.

1. Run the algorithm ALIGN(A, B).
2. If $d_F(A, B) > \delta_1 + \delta_2 + \delta_3$, report 'no valid solution' and exit.
3. Initialize $a'_1 \longleftarrow a_1, b'_1 \longleftarrow b_1, i \leftarrow 1, j \leftarrow 1$.
4. Loop until $i = j = K$.
   (a) Let $\langle a_{i,1}, a_{i,2}, ..., a_{i,p} (= a_I) \rangle$ be the maximal subsequence of A which is inside $\mathcal{B}(a'_i, \delta_1)$ and let $\langle b_{j,1}, b_{j,2}, ..., b_{j,q} (= b_J) \rangle$ be the maximal subsequence of B which is inside $\mathcal{B}(b'_j, \delta_2)$. (Note that $a'_i = a_{i,p'}$ for some $p' \leq p$ and $b'_j = b_{j,q'}$ for some $q' \leq q$.)
   (b) Let $\langle a_{I+1}, a_{I+2}, ..., a_{I+s} \rangle$ be the maximal subsequence of A which is inside $\mathcal{B}(a_{I+s'}, \delta_1)$ and let $\langle b_{J+1}, b_{J+2}, ..., b_{J+t} \rangle$ be the maximal subsequence of B which is inside $\mathcal{B}(b_{J+t'}, \delta_2)$, with $s' \leq s, t' \leq t$.
   (c) If $d(a_{I+s'}, b_{J+t'}) \leq \delta_3$, then
      i. $I \longleftarrow I + s$, $J \longleftarrow J + t$,
      ii. $a'_{i+1} \longleftarrow a_{I+s'}$, $b'_{j+1} \longleftarrow b_{J+t'}$,
      iii. $i \leftarrow i + 1, j \leftarrow j + 1$.
   (d) Else if $d(a'_i, b_{J+t'}) \leq \delta_3$, then
      i. $J \longleftarrow J + t$, $b'_{j+1} \longleftarrow b_{J+t'}$,
      ii. $j \leftarrow j + 1$.
   (e) Else if $d(a'_{I+s'}, b_j) \leq \delta_3$, then
      i. $I \longleftarrow I + s$, $a'_{i+1} \longleftarrow a_{I+s'}$,
      ii. $i \leftarrow i + 1$.
   (f) Else backtrack by successively letting a'_i be $a_{i,p'-1}, a_{i,p'-2}, ..., a_{i,1}$ and letting b'_j be $b_{j,q'-1}, b_{j,q'-2}, ..., b_{j,1}$, and loop over Steps (a) through (e). If neither i nor j can be incremented over these pairs of a'_i and b'_j, exit with a report 'no valid solution'.

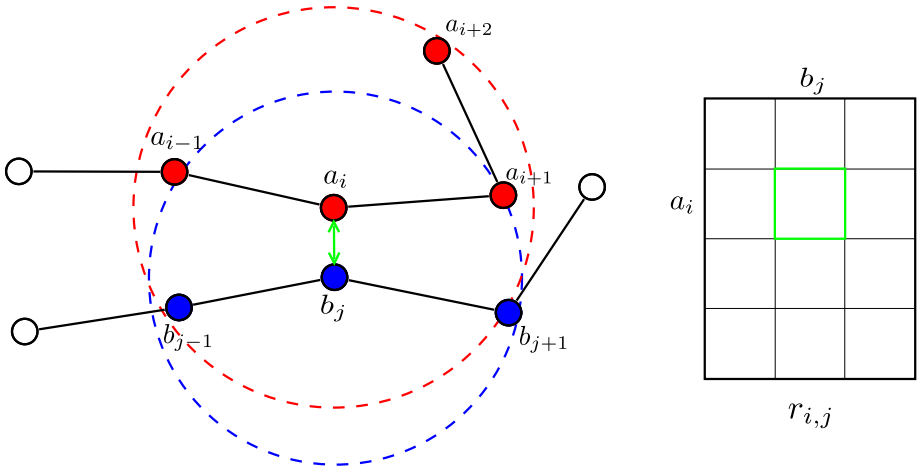


Figure 4.1: The rectangle $r_{i,j}$ constructed from subchains of A, B where $d(a_i, b_j) \leq \delta_3$. Here $S_A(a_i, \delta_1)$ contains the vertices a_{i-1} to a_{i+2}, and $S_B(b_j, \delta_2)$ contains the vertices b_{j-1} to b_{j+1}. Thus, $r_{i,j}$ is defined by the min and max node indices in each subchain.

4.3.1 CPS-3F$^+$ is Polynomial

In this section we present a polynomial time solution for CPS-3F$^+$. Several versions of the single chain simplification problem were addressed and shown to be polynomially solvable by Bereg et al. [7]. However, CPS-2H (where the Hausdorff distance is used for $d(A, A')$ and $d(B, B')$) was shown to be **NP**-complete and thus it is believed that the Fréchet version might be as well. The solution presented here proves that under the discrete Fréchet distance, the constrained chain pair simplification problem (CPS-3F$^+$) is polynomially solvable when the dimension is fixed. The algorithm returns the optimal K', specified in the definition of the decision problem, which is equal to

$$m_c(A', B') = \min_W m_c^W(A', B'), \tag{4.1}$$

among all feasible W. We now define several necessary terms and data structures.

Given two polygonal chains $A = \langle a_1, a_2, ..., a_m \rangle$, $B = \langle b_1, b_2, ..., b_n \rangle$, and constraints $\delta_1, \delta_2, \delta_3 \in \mathbb{R}^+$, we can design a dynamic programming algorithm to find the optimal moving cost K'. First we let $\mathcal{D} = \{(a_i, b_j) |\ a_i \in A,\ b_j \in B \text{ and } d(a_i, b_j) \leq \delta_3\}$. This is the set of all pairs of nodes between the two chains which are at a distance of at most δ_3 from each other. Then we can define a matrix $\mathcal{C}$ of size $m \times n$, which in any cell $\mathcal{C}_{i,j}$, contains the minimum number, K', of pairs $(a_k, b_l) \in \mathcal{D}$, which given δ_1, δ_2, and δ_3 simplify A and B via CPS-3F$^+$ from (a_1, b_1) up to (a_i, b_j).

In order to maintain $\mathcal{C}$, we need another data structure $\mathcal{R}$ and some other helpful definitions that we will refer back to later.

Definition 19. We define $S_X(x_i, \delta)$ as the maximal continuous subchain containing x_i on the polygonal chain X such that all the vertices on this subchain are contained in the sphere centered at x_i with radius δ.

Definition 20. Now let $r_{i,j}$ be the rectangle defined as $\langle \min(S_A(a_i, \delta_1)), \max(S_A(a_i, \delta_1)), \min(S_B(b_j, \delta_2)), \max(S_B(b_j, \delta_2)) \rangle$ such that $(a_i, b_j) \in \mathcal{D}$ (Figure 4.1). Here, min and max refer to the minimum or maximum indexed element within $S_X(x_i, \delta)$.

For every pair in $\mathcal{D}$, we envision the corresponding rectangles as being overlayed on $\mathcal{C}$. A rectangle $r_{i,j}$ covers all the cells of $\mathcal{C}$ that are analogous to the vertices in $S_A(a_i, \delta_1) \cup S_B(b_j, \delta_2)$ as shown in Figure 4.1.

Our dynamic programming approach must take two different concurrent simplifications into account to find the minimum moving cost. First, the distance between each original and simplified chain is represented by the sphere, $S_X(x_i, \delta)$, and captured as part of a rectangle (Figure 4.1). Second, $\mathcal{D}$ represents the distance between the two simplified chains. The two are combined by the rectangles where the height and width capture the simplified vertices in A and B, respectively, and the number and placement of those rectangles on $\mathcal{C}$ ties in the relationship between the two chains.

For convenience we also define the set of all rectangles that a cell in $\mathcal{C}$ belongs to: $\mathcal{Q}_{k,l} = \{r_{i,j} | a_k \in A, b_l \in B$ and $\min(S_A(a_i, \delta_1)) \leq a_k \leq \max(S_A(a_i, \delta_1))$ and $\min(S_B(b_j, \delta_2)) \leq b_l \leq \max(S_B(b_j, \delta_2))\}$.

Let $\mathcal{R}$ be a matrix of sets where the matrix is of size m by n, and $\mathcal{R}$ provides information needed to fill out $\mathcal{C}$ by storing a list of rectangles for each cell. $\mathcal{R}_{i,j}$ contains a set of rectangles (dynamic array) which pertain to the number of coverings (rectangles) still viable at any (i, j) relating to the number already calculated for $\mathcal{C}_{i,j}$. These are computed by the recurrences in Equations 4.2 and 4.3, which are shown along with the initial conditions for the relations.

Initial Conditions: $\mathcal{Q}_{1,1} \neq \emptyset$, $\mathcal{R}_{1,1} = \mathcal{Q}_{1,1}$, and $\mathcal{C}_{1,1} = 1$.

$$\mathcal{C}_{i,j} = \min_{(k,l)\in\{(i\text{-}1,j),(i,j\text{-}1),(i\text{-}1,j\text{-}1)\}} \begin{cases} \mathcal{C}_{k,l}, & \text{if } \mathcal{Q}_{i,j} \cap \mathcal{R}_{k,l} \neq \emptyset \\ \mathcal{C}_{k,l} + 1, & \text{if } \mathcal{Q}_{i,j} \cap \mathcal{R}_{k,l} = \emptyset,\ \mathcal{Q}_{i,j} \neq \emptyset,\ \mathcal{R}_{k,l} \neq \emptyset \\ NULL, & \text{if } \mathcal{Q}_{i,j} = \emptyset \end{cases} \tag{4.2}$$

$$\mathcal{R}_{i,j} = \bigcup_{(k,l)\in\{(i\text{-}1,j),(i,j\text{-}1),(i\text{-}1,j\text{-}1)\}} \begin{cases} \mathcal{R}_{k,l} \cap \mathcal{Q}_{i,j}, & \text{if } \mathcal{C}_{i,j} = \mathcal{C}_{k,l},\ \mathcal{R}_{k,l} \cap \mathcal{Q}_{i,j} \neq \emptyset \\ \mathcal{Q}_{i,j}, & \text{if } \mathcal{C}_{i,j} = \mathcal{C}_{k,l} + 1,\ \mathcal{R}_{k,l} \neq \emptyset, \\ & \mathcal{R}_{k,l} \cap \mathcal{Q}_{i,j} = \emptyset \end{cases} \tag{4.3}$$

The idea is to find the minimum covered xy-monotone increasing path from (a_1, b_1) to (a_m, b_n) which corresponds to $\mathcal{C}_{1,1}$ to $\mathcal{C}_{m,n}$. This is the minimum path by basic dynamic programming with all feasible options explored. If we visited a cell that was not covered, that would mean one of the nodes is not covered by a pair in $\mathcal{D}$. By finding a minimum covered path, one guarantees that every column and every row is covered by at least one rectangle which means all of the nodes of A and B are covered.

The increasing xy-monotone path is necessary in the recurrence due to the definition of the discrete Fréchet distance. Without the requirement of a monotonically

increasing path we would be using the weak discrete Fréchet distance – a version of the Fréchet distance that allows backtracking, or in terms of the analogy would allow the man or the dog to walk backwards.

We first characterize the optimal substructure of CPS-3F$^+$ as an optimization problem given our definitions, and then show this yields the optimal solution for K' and thus decides CPS-3F$^+$.

Theorem 2. *Optimal substructure of CPS-3F$^+$:*
Let $A = \langle a_1, \ldots, a_m \rangle$ and $B = \langle b_1, \ldots, b_n \rangle$ be two polygonal chains, $\delta_1, \delta_2, \delta_3 \in \mathbb{R}^+$, and let $Z_i = \langle z_1, \ldots, z_i \rangle$ be any CPS-3F$^+$ solution such that every z_j is a rectangle.

1. *If (a_k, b_l) is covered by z_i where $(k,l) \in \{(m\text{-}1, n), (m, n\text{-}1), (m\text{-}1, n\text{-}1)\}$, then Z_i is a CPS-3F$^+$ solution for A_k, B_l.*

2. *If (a_k, b_l) is covered by z_{i-1} where $(k,l) \in \{(m\text{-}1, n), (m, n\text{-}1), (m\text{-}1, n\text{-}1)\}$, then Z_{i-1} is a CPS-3F$^+$ solution for A_k, B_l.*

3. *If (a_k, b_l) is not covered by z_i or z_{i-1} where $(k,l) \in \{(m\text{-}1, n), (m, n\text{-}1), (m\text{-}1, n\text{-}1)\}$, then $\nexists$ a CPS-3F$^+$ solution for A_k, B_l.*

Proof. **1)** If z_i covers (a_k, b_l) where $(k,l) \in \{(m\text{-}1, n), (m, n\text{-}1), (m\text{-}1, n\text{-}1)\}$, then $z_i \in \mathcal{Q}_{m,n} \cap \mathcal{R}_{k,l}$, and $C_{k,l} = |Z_i|$. Suppose $C_{k,l} \neq |Z_i|$, then for (a_k, b_l) there are two possibilities: either $C_{k,l} > |Z_i|$ or $C_{k,l} < |Z_i|$. $C_{k,l} > |Z_i|$ implies the solution required another rectangle at the previous step, but since the recurrence is monotonically increasing this is impossible. If $C_{k,l} < |Z_i|$, then given the addition of z_i for (m,n) implies $z_i \notin \mathcal{Q}_{m,n} \cap \mathcal{R}_{k,l}$, but that contradicts our assumption.

2) If z_{i-1} covers (a_k, b_l), where $(k,l) \in \{(m\text{-}1, n), (m, n\text{-}1), (m\text{-}1, n\text{-}1)\}$, but not (a_m, b_n) then $z_i \notin \mathcal{Q}_{m,n} \cap \mathcal{R}_{k,l}$, and $C_{k,l} = |Z_{i-1}| = |Z_i|$-1. If we suppose $C_{k,l} \neq |Z_{i-1}|$, then either $C_{k,l} > |Z_{i-1}|$ or $C_{k,l} < |Z_{i-1}|$. If $C_{k,l} > |Z_{i-1}|$, then $C_{k,l} = |Z_{i-1}|+1 = |Z_i|$,

and since Z_i is an optimal solution we have added another rectangle that must cover (a_k, b_l) and (a_m, b_n) in order to be a solution. However, this means $\exists\ z_i \in \mathcal{Q}_{m,n} \cap \mathcal{R}_{k,l}$, which contradicts our assumption. Suppose $C_{k,l} < |Z_{i-1}|$, then to cover (a_m, b_n) we must add another rectangle, but that contradicts Z_i being an optimal solution since we have covered A_m, B_n with $|Z_i|$-1 rectangles.

3) Since $\nexists$ any rectangles z_i and z_{i-1} that cover both (a_m, b_n) and (a_k, b_l), where $(k, l) \in \{(m\text{-}1, n), (m, n\text{-}1), (m\text{-}1, n\text{-}1)\}$, then $\mathcal{Q}_{k,l} = \emptyset$ or $\mathcal{Q}_{m,n} = \emptyset$. By definition, if no rectangles cover the cells, then there is no solution for A_k, B_l or A_m, B_n. □

Theorem 3. *Constrained chain pair simplification, under the discrete Fréchet distance, is polynomially solvable, i.e. CPS-3F$^+$ $\in$ **P**.*

Proof. Since we have shown that CPS-3F$^+$ has an optimal substructure, given A, B, K', δ_1, δ_2, and δ_3, we can find an optimal K'' from our dynamic programming algorithm (Algorithm 4.3). Then we decide CPS-3F$^+$ by comparing whether $K' \leq K''$. □

Corollary 1. Constrained chain pair simplification gives a factor 2-approximation to the chain pair simplification problem under the discrete Fréchet distance, i.e., CPS-3F$^+$ provides a 2-approximation of CPS-3F.

Proof. Given two polygonal chains A, B, let K be an optimal solution from CPS-3F yielding the simplified chains A', B', and assume K' is an optimal solution for CPS-3F$^+$ yielding the simplified chains A'', B'', i.e. $K' = m_c(A'', B'')$.

Here, $K \leq K'$ because K is the minimum number of vertices possible and the moving cost for any pair A'', B'' is at least equal to $\max(|A''|, |B''|)$, where $K \leq |A''|$ or $K \leq |B''|$.

Now, we know that the moving cost for A', B' satisfies $m_c(A', B') \geq K'$ since K' is an optimal moving cost for A, B. Thus, a monotonically increasing moving cost in

A', B' is at most $|A'| + |B'| - t$ for a t-walk. Now, $K' = m_c(A'', B'') \leq m_c(A', B') \leq |A'| + |B'| - t \leq 2\max(|A'|, |B'|) = 2K$. Thus, $K \leq K' \leq 2K$. □

4.3.2 Algorithm Complexity

The time complexity of the algorithm is largely dependent on δ_1, δ_2, and δ_3 because they define the size and number of rectangles. We allow δ_1, δ_2, and δ_3 to be absorbed in the complexity because their values do not guarantee a specific number of rectangles to be considered, nor how large a given rectangle is.

We can easily bound the complexity between $O(mn)$ and $O(m^2n^2)$. If the values of δ_1, δ_2, and δ_3 are small then any cell will only have a small constant number of rectangles to consider and the algorithm runs in $O(mn)$ time, which is the case for most protein related data.

However, in the worst case, if δ_1, δ_2, and δ_3 are set larger than the lengths of the chains causing every $\mathcal{Q}_{i,j}$ to contain all possible rectangles between the two chains, then the complexity is $O(m^2n^2)$ given the largest $|\mathcal{D}|$ possible is mn. The optimal solution for CPS-3F$^+$ in this case is $K' = 1$, but would require $O(m^2n^2)$ time. δ_3 is the largest contributing variable to the running time because it determines the number of possible rectangles (size of $\mathcal{D}$).

Filtering steps could be added to watch for large δ values. With filtering, the time at any step could be kept to a constant (or logarithmic) value and the time complexity would remain close to $O(mn)$ or have an extra logarithmic factor dependent on δ_1, δ_2, and δ_3.

The space complexity also has similar bounds, requiring a minimum of $O(mn)$ space and a maximum of $O(m^2n^2)$ space, if $\mathcal{Q}$ is used naïvely and built beforehand. The recurrences themselves only require two rows of data for either $|A|$ or $|B|$, so the

space complexity is linear to the size of the smaller chain (WLOG $O(n)$). However, this would require calculating $\mathcal{Q}_{i,j}$ at every step for the cell, which as discussed, could be expensive if the δ values are large.

4.3.3 Counter Example

We now briefly give an example proving that finding a solution to CPS-3F does not imply a solution to CPS-3F$^+$, and that having all solutions of CPS-3F$^+$ does not guarantee that one of the solutions will have a minimum K for CPS-3F.

A	B
$a_1 = (0, 0.85, 2.4)$	$b_1 = (0, 2, 2.5)$
$(0, 1, 1.5)$	$(0, 2, 1.5)$
$(0.4, 1, 0.6)$	$(0, 2.5, 1)$
$(0.9, 1.1, 0)$	$(0, 1.95, 0.5)$
$(0.6, 0.6, -0.3)$	$(0, 1, 0.5)$
$(0.5, 0, 0)$	$(0, 0, 0.5)$
$(1.5, 0, 0)$	$(0.75, 0.15, 1.1)$
$(2.45, 0, 0)$	$(1, 1, 1.5)$
$(3, 0.5, 0)$	$(1.3, 0.5, 0.75)$
$(2.5, 1, 0)$	$(1.6, 0.9, 0)$
$a_{11} = (2.5, 2, 0)$	$b_{11} = (1.5, 1.8, -0.4)$

Table 4.1: The nodes for chains A and B in the counter-example.

The nodes of A and B are listed in Table 4.1, and Figure 4.2 shows the resulting polygonal chains in $\mathbb{R}^3$. The dotted lines show the nodes between A and B that are within δ_3 of each other. The settings for both CPS-3F and CPS-3F$^+$, given A, B, are $\delta_1 = \delta_2 = \delta_3 = 1$.

This example has only nine pairs of nodes that correlate to rectangles in the dynamic programming solution (dotted lines). There are two paths that can be taken which are covered by rectangles. The minimum moving cost is equal to five and uses

five nodes in the simplified chains, i.e. $K = 5$, $K' = 5$. The other path requires six pairs, or rectangles, to be used but only needs four vertices in each simplified chain, $K = 4$, $K' = 6$. Thus, a worse CPS-3F$^+$ solution (higher moving cost) may yield a better CPS-3F solution (lower number of vertices).

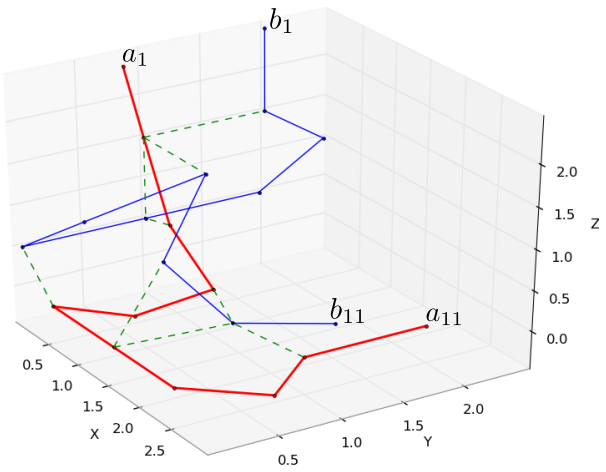


Figure 4.2: Two polygonal chains A and B. The dotted lines represent the nodes between A and B that are within δ_3 of each other.

In the next section we outline our algorithm and then walk through it using these example chains to demonstrate some of the intuition behind the algorithm. This explicitly shows that two paths are possible, and why we must choose which one is optimal based on the moving cost.

4.3.4 Dynamic Programming Algorithm

In this section we use the recurrences as a basis to find the optimal solution for a given set of inputs. Once the optimal solution is calculated we can easily decide CPS-3F$^+$. The algorithms presented assume many global or class variables outside of the functions. These variables are explained momentarily.

Since the recurrences only require the previous row and column for any decision, the function can be implemented as a simple iterative algorithm as opposed to a recursive one. Similar to the edit distance problem between two strings, only two rows at a time are needed, so the amount of memory can be reduced by only storing the two rows of $\mathcal{C}$ and $\mathcal{R}$ that the algorithm is currently using. This approach gives the optimal K' value, but in order to retrieve the actual K' pairs (via Algorithm 4.4), all rows of $\mathcal{C}$ and $\mathcal{R}$ must be stored.

Algorithm 4.3 assumes that $\mathcal{C}$ is an $m \times n$ matrix, where m, n are the sizes of A, B, respectively, and every cell is initialized to NULL. $\mathcal{R}$ is an $m \times n$ dynamic 3D jagged array where every element, $\mathcal{R}_{i,j}$, is its own array. We assume that $\mathcal{Q}$ has been calculated before this function is called. $\mathcal{Q}$ requires all rectangles to be enumerated, and for lower and upper bounds for each one to be set. Since this may vary between cells, $\mathcal{Q}$ is also a 3D jagged array like $\mathcal{R}$, and this makes assignments between the two easier.

For simplicity, to dynamically append to the end of an array or create a new row we use a generic function 'ADD' that takes the data structure as the first argument, and the value to append as the second. The method by which it works is context-sensitive to the type of data structure.

Algorithm 4.4 defines a recursive function that finds the rectangles through which the optimal path exists. The 'Cost' variable represents the pairs of nodes used in the minimum moving cost, and thus begins as the optimal K' value ($\mathcal{C}_{m,n}$) when the function is originally called. 'i' and 'j' are originally set to $|A|$ and $|B|$ respectively, and 'CurrRect' is set to NULL. The method assumes a jagged array '$Path$' of size K' that must exist to store the rectangles of the path. The idea is that we know our optimal path is of size K', and thus we store all possible paths of that length in our

Algorithm 4.3 FIND-CPS-3F$^+$ $\rightarrow$ Return the optimal moving cost, K', iteratively.
Output: The optimal moving cost between the simplified A and B.

```
1: function FIND-CPS-3F+
2:     for i ← 1, |A| do
3:         for j ← 1, |B| do
4:             if i = 1 and j = 1 then
5:                 C_{1,1} ← 1
6:                 R_{1,1} ← Q_{1,1}
7:             else
8:                 Values, Rectangles ← Array
9:                 for each (k, l) ∈ {(i-1, j), (i, j-1), (i-1, j-1)} do
10:                    if k > 0 and l > 0 then
11:                        for each rect ∈ R_{k,l} do
12:                            if (i, j) ∈ rect then
13:                                ADD(Values, C_{k,l} )
14:                                ADD(Rectangles, rect )
15:                            else
16:                                ADD(Values, C_{k,l} + 1 )
17:                                ADD(Rectangles, NULL )
18:
19:                C_{i,j} ← min( Values )
20:                for r ← 1, |Values| do
21:                    if Values[ r ] = C_{i,j} then
22:                        if Rectangles[ r ] = NULL then
23:                            ADD(R_{i,j}, Q_{i,j} \ R_{i,j} )
24:                        else
25:                            ADD(R_{i,j}, Rectangles[ r ] )
26:    return C_{m,n}
```

Path variable. 'CurrRect' simply refers to the current rectangle that the function is traversing.

This method does not tell us which rectangles connect with the rectangles at the next level. Therefore, in an actual implementation these stored rectangles should also have child or parent pointers to make it easier to follow the path. The pointers have been omitted here for algorithm clarity. An alternative to having pointers is to add

Algorithm 4.4 GET-PATH-CPS-3F$^+$ $\rightarrow$ Return all simplified paths between the chains of optimal moving cost length.

Input: i, j are indices of $a_i \in A, b_j \in B$. Cost is the current moving cost, and CurrRect is the current rectangle being evaluated.

```
1: procedure GET-PATH-CPS-3F+(i, j, Cost, CurrRect)
2:     if CurrRect = NULL then
3:         for each rect ∈ R_{i,j} do
4:             ADD(Path[ Cost ], rect )
5:             GET-PATH-CPS-3F+( i, j, Cost, rect )
6:     for each (k, l) ∈ {(i-1, j), (i, j-1), (i-1, j-1)} do
7:         if k > 0 and l > 0 and C_{k,l} != NULL then
8:             if C_{k,l} = C_{i,j} then
9:                 GET-PATH-CPS-3F+( k, l, Cost, CurrRect )
10:            else if C_{k,l} = C_{i,j} − 1 then
11:                for each rect ∈ R_{k,l} do
12:                    if rect ∉ Path[ Cost ] then
13:                        if rect ∉ Path[ Cost - 1 ] then
14:                            ADD(Path[ Cost - 1 ], rect )
15:                        GET-PATH-CPS-3F+( k, l, Cost - 1, rect )
```

an 'if' statement to check if $Path[1]$ has a value and then exit if it does. This means at least one optimal path has been found (there could be multiple).

4.3.5 Example Walkthrough

Here, we briefly walk through parts of Algorithm 4.3 for the example chains listed in Subsection 4.3.3. We let $\delta_1 = \delta_2 = \delta_3 = 1$ for our given chains A and B. Subsequently, we get nine pairs in $\mathcal{D} = \{(a_2, b_2),(a_2, b_5),(a_2, b_8),(a_3, b_5),(a_4, b_{10}),(a_6, b_6),(a_7, b_9),(a_7, b_{10}),(a_{10}, b_{10})\}$, which are each represented by a rectangle. This example was designed so that each rectangle only covers its nearest neighbors in the chain. Thus, for example, (a_3, b_5) covers all nodes in our $A \times B$ grid from (a_2, b_4) to (a_4, b_6).

There are only two paths through the rectangles. The path $p_1 = \langle r_{2,2}, r_{3,5}, r_{6,6}, r_{7,9}, r_{10,10} \rangle$ and the path $p_2 = \langle r_{2,2}, r_{2,5}, r_{2,8}, r_{4,10}, r_{7,10}, r_{10,10} \rangle$. When we begin $\mathcal{C}_{1,1} = 1$ and $\mathcal{R}_{1,1} = r_{2,2}$. When the path leaves the rectangle $r_{2,2}$ is at $\mathcal{C}_{k,3}$, where $k \in \{1, 2, 3\}$. For $k = 1$ the only rectangle to choose is $r_{2,5}$, but with $k = 2$ or $k = 3$ the corresponding $\mathcal{R}_{k,3} = \{r_{2,5}, r_{3,5}\}$.

The two paths then diverge and are straightforward until we evaluate $\mathcal{C}_{9,9}$ when we face the choice of which path is "better". Looking at the values of the previous three squares $\mathcal{C}_{8,9}, \mathcal{C}_{9,8}$, and $\mathcal{C}_{8,8}$, the path p_1 has fewer rectangles (four) compared to p_2, which has five, and thus we set $\mathcal{C}_{9,9} = 4$ and $\mathcal{R}_{9,9} = r_{7,9}$. The subsequent steps will all be equal, $\mathcal{C}_{9,10} = \mathcal{C}_{10,9} = \mathcal{C}_{10,10} = 5$, and $\mathcal{R}_{9,10} = \mathcal{R}_{10,9} = \mathcal{R}_{10,10} = r_{10,10}$ because the algorithm leaves rectangle $r_{7,9}$ and is now in the only rectangle left.

When we get to $\mathcal{C}_{11,11}$ the value remains at five because we are still covered by $r_{10,10}$, and thus we return five as the moving cost. Running Algorithm 4.4 returns the pairs of vertices that comprise the path p_1, which is $\langle (a_2, b_2), (a_3, b_5), (a_6, b_6), (a_7, b_9), (a_{10}, b_{10}) \rangle$.

4.4 Empirical Results

We now present some results comparing our previous heuristic method SIMPLIFY (Algorithm 4.2) and the 2-approximation solution (Algorithms 4.3 and 4.4) of CPS-3F$^+$. We present the results for chains with a similar length and then consider dissimilar chains of various lengths in order to vary the amount of simplification per chain. The algorithms were implemented in both Python and C# and more information about the FPACT software is available in Chapter 8. On an older 32 bit quad-core machine the algorithm took anywhere from a few seconds to several minutes depending on the parameters. The long runs were for simplifications in which δ_1, δ_2, and δ_3 were all set to large values (Tables 4.3 and 4.4).

Protein Chain	$\lvert B\rvert$	$d_F(A,B)$	δ_1	δ_2	δ_3	$\lvert A'\rvert$	$\lvert B'\rvert$	K'	$d_F(A',B')$	Algorithm
1hfj.c	325	0.95	4	4	1	109	109	109	0.95	SIMPLIFY
					1	**109**	**109**	**109**	**0.95**	CPS-3F$^+$
1qd1.b	325	22.65	4	4	47	110	109	110	24.96	SIMPLIFY
					21	**117**	**126**	**150**	**20.70**	CPS-3F$^+$
1toh	325	22.06	4	4	60	109	110	110	23.39	SIMPLIFY
					21	**149**	**130**	**178**	**20.54**	CPS-3F$^+$
4eca.c	325	5.55	4	4	20	109	109	109	7.96	SIMPLIFY
					6	**110**	**111**	**111**	**5.97**	CPS-3F$^+$
1d9q.d	297	20.87	4	4	43	109	108	109	23.68	SIMPLIFY
					20	**130**	**127**	**166**	**19.86**	CPS-3F$^+$
4eca.b	325	5.64	4	4	17	109	109	109	7.51	SIMPLIFY
					5	**110**	**111**	**111**	**4.89**	CPS-3F$^+$
4eca.d	325	5.71	4	4	18	109	109	109	7.82	SIMPLIFY
					5	**111**	**113**	**113**	**4.94**	CPS-3F$^+$

Table 4.2: Comparison of Algorithm SIMPLIFY (Algorithm 4.2) and FIND-CPS-3F$^+$ (Algorithm 4.3) with 107j.a (Chain A) of length 325. $\delta_1 = \delta_2 = 4$, and δ_3 set to the minimal value. The heuristic method is in [2] and the CPS-3F$^+$ results are in [3].

We note that in the result sections, the RMSD values were taken from ProteinDBS, and thus the alignment length, or coverage, is not the full length of each chain [34]. This is especially true when discussing chains of different lengths in 4.4. This makes a straightforward comparison between the chains using both RMSD and the discrete Fréchet distance difficult. However, the results are mainly to compare CPS-3F$^+$ to our previous algorithm SIMPLIFY (Algorithm 4.2), and thus the coverage is not listed.

4.4.1 Similar Chain Length Comparisons

Using the same format as our previous results, we set $\delta_1 = \delta_2$ for simplicity and to ensure chains A', B' will have a similar reduced length since nearly all are the

same length initially. δ_3 is set to the minimum integer value that will reduce the chains via CPS-3F$^+$ given δ_1, δ_2. The comparison tables in both cases are using the protein backbone 107j.a (protein A) and comparing it with seven other chains from the Protein DataBank: 1hfj.c, 1qd1.b, 1toh, 4eca.c, 1d9q.d, 4eca.b, 4eca.d. These seven chains were reported to be similar to 107j.a by the ProteinDBS software [34] (this took a few seconds searching the whole PDB, which contained over 30,000 protein backbones at that time). Previously, [1] used a heuristic algorithm based on the discrete Fréchet distance and showed that three of the seven chains were not actually similar to 107j.a, and ProteinDBS has subsequently updated their page to reflect this. 107j.a has 325 nodes along the backbone and all but one of the seven other chains do as well.

For the CPS-3F$^+$ algorithm, all chains are assumed to be aligned, and we use the alignments from our previous algorithm ALIGN [2]. In Table 4.2 we fixed $\delta_1 = \delta_2 = 4$ since the distance between two α-carbon atoms in the backbone is approximately $\approx$ 3.7 to 3.8 (Ångströms). This value ensures that we will be simplifying the chains a minimal amount. We can see that we get an approximate reduced length of 1/3 which is what we would expect (since this distance will only use the neighboring nodes). The optimal algorithm allows for δ_3 to be much smaller than the heuristic because it can simplify the chains with a value often less than $d_F(A, B)$, and hence $d_F(A', B')$ is a lower value.

In Table 4.3 we vary δ_1 and δ_2 for different amounts of simplification and again set δ_3 to the minimum integer value that allows for simplification via CPS-3F$^+$. We keep $\delta_1 = \delta_2$ for simplicity and to ensure a similar reduced size for both chains. Here we have a more dramatic difference in δ_3 and in $d_F(A', B')$ because of the greater simplification possibilities between A, A' and B, B' since δ_1, δ_2 are much larger. This demonstrates how CPS-3F$^+$ is able to simplify the two chains simultaneously while

Protein Chain	$\lvert B\rvert$	$d_F(A,B)$	δ_1	δ_2	δ_3	$\lvert A'\rvert$	$\lvert B'\rvert$	K'	$d_F(A',B')$	Algorithm
1hfj.c	325	0.95	12	12	4	28	28	28	3.77	SIMPLIFY
					1	**26**	**26**	**26**	**0.95**	CPS-3F$^+$
1qd1.b	325	22.65	15	15	33	16	17	17	22.64	SIMPLIFY
					12	**21**	**23**	**24**	**11.94**	CPS-3F$^+$
1toh	325	22.06	16	16	34	18	16	18	28.51	SIMPLIFY
					13	**22**	**19**	**22**	**12.80**	CPS-3F$^+$
4eca.c	325	5.55	12	12	12	28	28	28	11.73	SIMPLIFY
					3	**27**	**27**	**27**	**2.90**	CPS-3F$^+$
1d9q.d	297	20.87	15	15	27	19	21	21	23.73	SIMPLIFY
					13	**22**	**24**	**26**	**12.99**	CPS-3F$^+$
4eca.b	325	5.64	12	12	8	28	28	28	7.81	SIMPLIFY
					3	**26**	**26**	**26**	**2.94**	CPS-3F$^+$
4eca.d	325	5.71	12	12	11	28	28	28	10.01	SIMPLIFY
					3	**32**	**32**	**32**	**2.99**	CPS-3F$^+$

Table 4.3: Comparison of Algorithm SIMPLIFY (Algorithm 4.2) and FIND-CPS-3F$^+$ (Algorithm 4.3) with 107j.a (Chain A) of length 325. $\delta_1 = \delta_2$, and δ_3 set to the minimal value. The heuristic method is in [2] and the CPS-3F$^+$ results are in [3].

highlighting the similarities between the two chains. This is especially noticeable in that the discrete Fréchet distance between the simplified chains, $d_F(A', B')$, is drastically less than that of the original chains, $d_F(A, B)$.

We can see that the optimal results far exceed the heuristic approximation. If we look at 4eca.c in Table 4.3, the difference between the heuristic (11.73) and the optimal (2.90) is dramatic. The optimal δ_3 for CPS-3F$^+$ is 3 to 4 times smaller than the heuristic in general, and the discrete Fréchet distance between A' and B' is smaller than the original distance between A, B.

The heuristic algorithm only allowed for a constant number of backtracking steps which resulted in both chains being simplified to a similar number of vertices. With

Protein Chain	$\lvert B\rvert$	$d_F(A,B)$	δ_1	δ_2	δ_3	$\lvert A'\rvert$	$\lvert B'\rvert$	K'	$d_F(A',B')$	Algorithm
3ntx.a	322	10.04	10	10	22	35	40	40	16.21	SIMPLIFY
					5	**39**	**39**	**39**	**4.91**	CPS-3F$^+$
1wls.a	316	11.97	15	13	32	16	25	25	20.50	SIMPLIFY
					6	**22**	**22**	**22**	**5.99**	CPS-3F$^+$
2eq5.a	215	22.35	8	6	39	53	43	53	23.47	SIMPLIFY
					19	**58**	**53**	**66**	**18.91**	CPS-3F$^+$
2zsk.a	219	21.92	12	8	30	27	31	31	24.60	SIMPLIFY
					17	**38**	**34**	**43**	**16.90**	CPS-3F$^+$
1zq1.a	363	23.38	10	12	40	36	37	37	28.30	SIMPLIFY
					19	**51**	**53**	**56**	**18.47**	CPS-3F$^+$
3jq0.a	457	27.36	6	9	52	71	54	71	30.75	SIMPLIFY
					26	**65**	**70**	**80**	**25.67**	CPS-3F$^+$
2fep.a	273	24.55	20	17	27	13	13	13	25.00	SIMPLIFY
					10	**10**	**11**	**11**	**9.94**	CPS-3F$^+$

Table 4.4: Comparison of Algorithm SIMPLIFY (Algorithm 4.2) and FIND-CPS-3F$^+$(Algorithm 4.3) with 107j.a (Chain A) of length 325, and various δ_1, δ_2, and δ_3 set to simplify both chains to a similar length. The heuristic method is in [2] and the CPS-3F$^+$ results are in [3].

CPS-3F$^+$, we can see that the chains can vary greatly in the amount they simplify in order to have a minimum moving cost.

4.4.2 Varying Chain Length Comparisons

One aspect of chain pair simplification we have not exploited is simplifying the chains differently. Here, we look at chains that vary in length, are not aligned as well with the base chain, and subsequently have a large discrete Fréchet distance. Table 4.4 shows these results. The values for δ_1 and δ_2 were chosen in an attempt to simplify both chains to a similar size via CPS-3F$^+$. This allows us to pull out the similarities of two chains that may be vastly different without simplification, yet still have some

subset of nodes that align and compare well. For visualization purposes, it lets us see the overall subset similarity structure of the two chains. This method could prove useful when finding nodes in each chain that match well together, i.e. they have a low moving cost and small discrete Fréchet distance.

The heuristic method SIMPLIFY (Algorithm 4.2) does not find similar optimal simplifications, and also ends up with a much higher moving cost and δ_3. The discrete Fréchet distance, consequently, is also much higher. As in our previous results, for both SIMPLIFY and CPS-3F$^+$, we picked δ_1 and δ_2, and then report the smallest integer value of δ_3 that worked for the respective algorithm.

The disparity between the number of vertices and the moving cost (K') is lessened if the chains simplify in a similar fashion. When δ_1 and δ_2 are large, but δ_3 is small, it allows for these larger "hops" to be made, and thus the simplified chains are similar in length, and the moving cost is a closer approximation to K. Using larger than minimum values for δ_3, we allow for greater flexibility in the simplification and it will yield a lower moving cost.

4.4.3 Visual Examples

In order to understand the benefits of using CPS-3F for visualization purposes, we briefly look at two examples. Figure 4.3 shows the alignment and simplification between 1o7j.a and 1hfj.c in three stages of simplification. They are unsimplified in (a) at 325 nodes each, they are simplified in (b) to 109 nodes, and further simplified to 26 nodes each in (c). These are the simplifications reported in Tables 4.2 and 4.3 between 1o7j.a and 1hfj.c using CPS-3F$^+$. In (a) the structures of the protein backbones are almost impossible to see, however, in (b) and (c) it becomes easy to see how the backbones move in the 3-dimensional space.

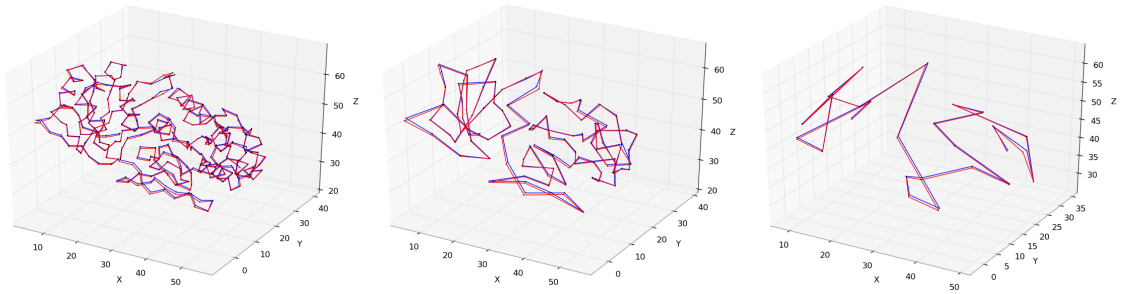
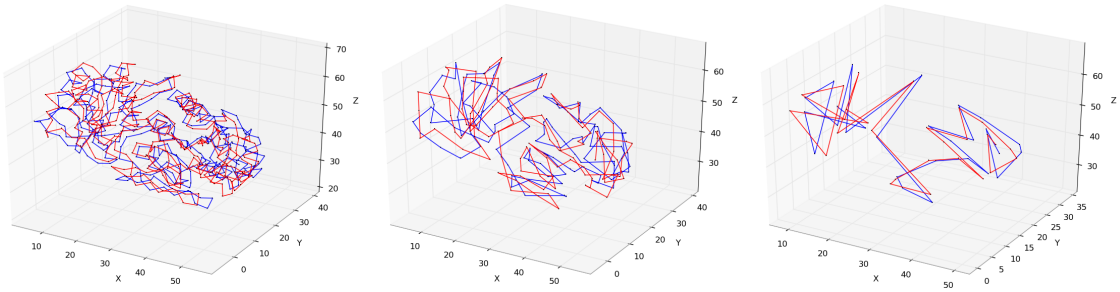


Figure 4.3: The alignment and simplification of 1o7j.a and 1hfj.c. (a) The alignment with no simplification with each chain having 325 nodes. (b) The chains simplified to 109 nodes each. (c) The chains simplified to only 26 nodes each.

Figure 4.4 shows the same basic levels of simplification between 1o7j.a and 4eca.c. Similarly, these are the simplifications shown in Tables 4.2 and 4.3 between 1o7j.a and 4eca.c using CPS-3F$^+$. The chain 4eca.c is not as similar to 1o7j.a as 1hfj.c, so their alignment is even more difficult to see with the unsimplified chains in (a). Even with only 110 and 111 nodes in (b) their similarity is hard to follow. In (c) though, with only 27 nodes in each chain, we can see the major movements of both chains in the space and where they are the most similar and dissimilar.

These two examples provide support for the benefits of CPS-3F in a visualization context. Further, it shows there may be benefit in comparing protein backbones by successive levels of simplification beginning with coarse representations and moving to more detailed comparisons. This allows researchers to understand how the two proteins behave in the space and then to fill out the models with more details. This also allows them to see the nodes that are better aligned between the two chains and where the chains align poorly.



(a) (b) (c)

Figure 4.4: The alignment and simplification of 1o7j.a and 4eca.c. (a) The alignment with no simplification with each chain having 325 nodes. (b) The chains simplified to $|A'| = 110$ and $|B'| = 111$ nodes. (c) The chains simplified to only 27 nodes each.

4.5 CPS-2H$^+$

4.5.1 CPS-2H$^+$ is NP-complete

We now prove the complexity of CPS-2H$^+$ by a reduction from 3-SAT. Although they were not dealing with a geometric problem, the basic idea of this proof comes from [37].

Theorem 4. *CPS-2H$^+$ is **NP**-complete.*

Proof. First, CPS-2H$^+$ is in **NP** since the three distance conditions and the moving cost can all be verified in polynomial time.

For 3-SAT we are given a set of boolean variables $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$ and a set of clauses $\mathcal{C} = \{c_1, c_2, \ldots, c_M\}$ where each $c = (v_i \vee v_j \vee v_k)$ such that $v_i, v_j, v_k \in \mathcal{X} \vee \neg\mathcal{X}$. We assert that any clause may not contain both x_i and $\neg x_i$ since that clause would be true and can be discarded from our construction, and that either x_i or $\neg x_i$ must

be in at least one clause. The problem is whether $\varphi = c_1 \wedge c_2 \wedge \cdots \wedge c_M$ in conjunctive normal form is satisfiable.

Since each clause, c_i, has three variables, WLOG we can enumerate them as c_i^k where $k \in \{0, 1, 2\}$. We can now create a point for each numbered variable k in clause c_i as $p_{i,k} = (i, i^2, k \cdot \varepsilon)$ where $k \in \{0, 1, 2\}$ and $0 < \varepsilon < 0.5$. Now we construct two polygonal chains A and B each with $(3M+N)$ vertices, such that φ is satisfiable $\iff$ A, B can be simplified into A', B', respectively, and $d_H(A, A') \leq 2\varepsilon$, $d_H(B, B') \leq 2\varepsilon$, $d_F(A', B') = 0$, $|A'| = |B'| = M + N$ and the moving cost $K' = M + N$. Given our construction, K' is equivalent to the number of vertices in each chain and the number of walks in the Fréchet alignment.

To begin we define a sequence of N points $q_j = (0, j, 10)$, where $1 \leq j \leq N$, as separators. Now for each $x_i \in \mathcal{X}$, we construct two sequences S_i and S_i^*. Let $c_{i_1}, \ldots, c_{i_u}$ be the sequence of clauses that contain x_i, and $c_{j_1}, \ldots, c_{j_v}$ the clauses containing $\neg x_i$. $S_i = \langle c_{i_1}, \ldots, c_{i_u}, c_{j_1}, \ldots, c_{j_v} \rangle$ and $S_i^* = \langle c_{j_1}, \ldots, c_{j_v}, c_{i_1}, \ldots, c_{i_u} \rangle$. Note that a clause c_i appears exactly three times in $\cup_{j=1}^{N} S_j$ and $\cup_{j=1}^{N} S_j^*$ since there are three literals in each clause, and we enumerate those literals as c_i^k where $k \in \{0, 1, 2\}$. Now, for each literal $x_i \in \mathcal{X}$, we convert S_i, S_i^* into the sequences T_i, T_i^* where every clause label is replaced by the point $p_{j,k}$ where $k \in \{0, 1, 2\}$ and k corresponds to the enumerated c_j^k in the clause of variable x_i. Since the order of the enumeration of the $p_{j,k}$ points is arbitrary in a clause, we will assume that they always occur in order $(p_{j,0}, p_{j,1}, p_{j,2})$.

Let $A = \langle T_1, q_1, T_2, q_2, \ldots, T_N, q_N \rangle$ and $B = \langle T_1^*, q_1, T_2^*, q_2, \ldots, T_N^*, q_N \rangle$.

We first show the forward direction. Suppose there exists a sequence of boolean assignments $\mathcal{Z} = \langle z_1, z_2, \ldots, z_N \rangle$ such that $x_i = z_i$ $(1 \leq i \leq N)$ satisfies φ. If $z_i = 1$, then T_i and T_i^* simplify to $T_i' = p_{i_1}, \ldots, p_{i_u}$ and $T_i^{*'} = p_{i_1}, \ldots, p_{i_u}$ respectively. However, if $z_i = 0$ we simplify both to the other sequence, $T_i' = p_{j_1}, \ldots, p_{j_v}$ and

$T_i^{*'} = p_{j_1}, \ldots, p_{j_v}$. To make $d_H(A, A') \leq 2\varepsilon$, $d_H(B, B') \leq 2\varepsilon$, $d_F(A', B') = 0$, and $|A'| = |B'| = M + N$, $K' = M + N$, we keep exactly one point among each triple of points $p_{j,k}$ ($k \in \{0, 1, 2\}$), and remove the remaining ones. Then, after the deletion, $A' = \langle T_1', q_1, T_2', q_2, \ldots, T_N', q_N \rangle$ is equivalent to $B' = \langle T_1^{*'}, q_1, T_2^{*'}, q_2, \ldots, T_N^{*'}, q_N \rangle$. The reason that $d_H(A, A') \leq 2\varepsilon$ and $d_H(B, B') \leq 2\varepsilon$ is because each point selected in $p_{j,k}$, $k \in \{0, 1, 2\}$, is at most a distance of 2ε from the removed points in the same triple (clause).

We now prove the opposite direction. Suppose that A, B are simplified into A'', B'', respectively, by removing some points in $\{p_{i,j} | 1 \leq i \leq M, 0 \leq j \leq 2\}$ such that $d_H(A, A'') \leq 2\varepsilon$, $d_H(B, B'') \leq 2\varepsilon$, $d_F(A'', B'') = 0$, and $|A''| = |B''| = K' = M + N$. We know $d(p_{i,k}, p_{j,l}) > 1$ and $d(q_i, q_j) \geq 1$ where $i \neq j$. The conditions $d_H(A, A'') \leq 2\varepsilon$ and $d_H(B, B'') \leq 2\varepsilon$ imply that we can only remove points in $\{p_{i,j} | 1 \leq i \leq M, 0 \leq j \leq 2\}$ while leaving at least one point in each triple for A'', B'' for each clause, i.e. one $p_{i,k}$, $0 \leq k \leq 2$. Since c_i can not contain both x_j and $\neg x_j$, there will be only one point p_a, for some a, on the subchain between q_r and q_{r+1} on A or B. The condition that $d_F(A', B') = 0$ implies that in A'' and B'' we must keep the same point among the triple $\{p_{i,j} | 1 \leq i \leq M, 0 \leq j \leq 2\}$. Finally, since $|A''| = |B''| = K' = M + N$, to make $d_F(A', B') = 0$ we must use all separator points (q's).

Let T_i'' and $T_i^{*''}$ be the subchains in A'' and B'' obtained from simplifying T_i and T_i^* in A and B, respectively. If T_i'' is empty, we can arbitrarily assign a value to x_i. However, if T_i'' is not empty, this implies that T_i'' and $T_i^{*''}$ have the same size and $d_F(T_i'', T_i^{*''}) = 0$. If T_i'' is not empty and it is a subsequence of $p_{i_1}, \ldots, p_{i_u}$, then we assign $x_i = 1$. If T_i'' is not empty and it is a subsequence of $p_{j_1}, \ldots, p_{j_v}$, then we assign $x_i = 0$. Thus, we can see that φ is satisfied by the assignments to the variables $x_i \in \mathcal{X}$.

Finally, we note that this reduction is polynomial and takes linear time based on the length of φ. □

4.5.2 Examples

Let $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$, $\mathcal{C} = \{c_1, c_2, c_3, c_4\}$, and $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$ where the clauses are assumed to be labeled in order. Thus, $N = 4$ and $M = 4$.

Now we construct the sequences. $S_1 = \langle c_1, c_2 \rangle$, $S_1^* = \langle c_2, c_1 \rangle$, $S_2 = \langle c_4, c_1, c_3 \rangle$, $S_2^* = \langle c_1, c_3, c_4 \rangle$, $S_3 = \langle c_1, c_2, c_3, c_4 \rangle$, $S_3^* = \langle c_2, c_3, c_4, c_1 \rangle$, $S_4 = \langle c_3, c_2, c_4 \rangle$, and $S_4^* = \langle c_2, c_4, c_3 \rangle$. The conversion to the T_i, T_i^* sequences merely replaces the label of the clause c_j with the point $p_{j,k}$ such that $k \in \{0, 1, 2\}$ is the place of variable x_i. For readability, we omit the comma in the subscript. $T_1 = \langle p_{10}, p_{20} \rangle$, $T_1^* = \langle p_{20}, p_{10} \rangle$, $T_2 = \langle p_{40}, p_{11}, p_{30} \rangle$, $T_2^* = \langle p_{11}, p_{30}, p_{40} \rangle$, $T_3 = \langle p_{12}, p_{21}, p_{31}, p_{41} \rangle$, $T_3^* = \langle p_{21}, p_{31}, p_{41}, p_{12} \rangle$, $T_4 = \langle p_{32}, p_{22}, p_{42} \rangle$, and $T_4^* = \langle p_{22}, p_{42}, p_{32} \rangle$.

Then we construct $A = \langle p_{10}, p_{20}, q_1, p_{40}, p_{11}, p_{30}, q_2, p_{12}, p_{21}, p_{31}, p_{41}, q_3, p_{32}, p_{22}, p_{42}, q_4 \rangle$ and $B = \langle p_{20}, p_{10}, q_1, p_{11}, \ p_{30}, p_{40}, q_2, p_{21}, p_{31}, p_{41}, p_{12}, q_3, p_{22}, p_{42}, p_{32}, q_4 \rangle$ where all points $q_j = (0, j, 10)$ for $j = 1, 2, 3, 4$. Note that $|A| = |B| = K' = 3M + N = 16$.

Suppose we set $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$. Then $A' = B' = \langle p_{10}, q_1, p_{40}, q_2, p_{31}, q_3, p_{22}, q_4 \rangle$ after we remove points where we already had a point from the triple. Thus, there are several possible simplified chains, depending on which points are removed. This gives $|A| = |B| = K' = M + N = 8$. Also notice that it is not necessary that the separating points and clauses (p_{ik}'s) are visited in order.

Figure 4.5 shows the points in the construction as well as one possible simplification. Each p_{ik} represents three points that have z values 0, ε, and 2ε for the clause c_i.

Notice this does not visit the nodes in the same order as our example, but is easier to see the general idea.

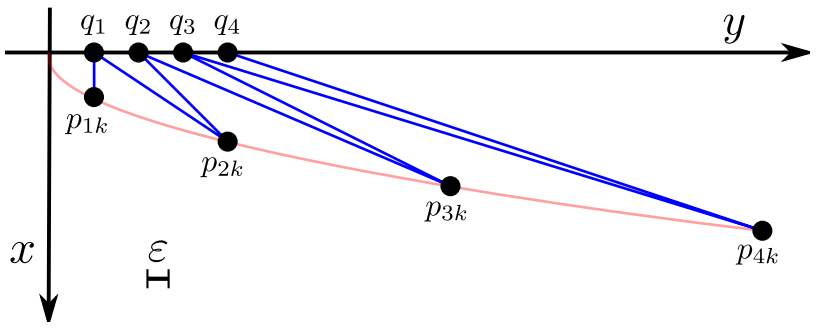


Figure 4.5: CPS-2H$^+$ reduction example viewed from the positive z axis. Each p_{ik} represents the three points for clause c_i which vary only in z. The line through only the p_{ik} points is $y = x^2$ to show where the clause points are placed.

Now we show a simple example that is not satisfiable. Let $\varphi = (x_1 \vee x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1 \vee \neg x_1)$. $S_1 = \langle c_1, c_1, c_1, c_2, c_2, c_2 \rangle$ and $S_1^* = \langle c_2, c_2, c_2, c_1, c_1, c_1 \rangle$. Then $T_1 = \langle p_{10}, p_{11}, p_{12}, p_{20}, p_{21}, p_{22} \rangle$ and $T_1^* = \langle p_{20}, p_{21}, p_{22}, p_{10}, p_{11}, p_{12} \rangle$.

Now $A = \langle p_{10}, p_{11}, p_{12}, p_{20}, p_{21}, p_{22}, q_1 \rangle$ and $B = \langle p_{20}, p_{21}, p_{22}, p_{10}, p_{11}, p_{12}, q_1 \rangle$. We end up with almost identical chains whether x_1 is 0 or 1. Let $x_1 = 1$, then $A' = \langle p_{10}, q_1 \rangle$ and $B' = \langle p_{10}, q_1 \rangle$ (any of the p_{1k}'s could have been chosen). Since none of the points from clause c_2 are in the simplified chains, $d_H(A, A') > 2\varepsilon$ and $d_H(B, B') > 2\varepsilon$ and thus it is not a valid CPS-2H$^+$ solution. The same result happens should $x_1 = 0$. Any solution will require $|A| = |B| = K' \geq M + N + 1$ and thus violate our reduction condition, meaning φ was not satisfiable.

CHAPTER 5

OTHER APPROACHES TO CHAIN PAIR SIMPLIFICATION

The moving cost is one measure that we are able to use to construct a 2-approximation for the chain pair simplification problem under the discrete Fréchet distance. Here, we look to extend that discussion with other measures of interest or other ways to approach the problem. We begin with a discussion of some of these properties and why they are useful.

5.1 Building Bridges

The discrete Fréchet distance is used in many areas where polygonal chain data is aligned, compared, and simplified such as protein data, computer graphics, etc. Here, we look to further understand the discrete Fréchet distance from a more theoretical aspect by exploiting its properties through the use of the Chain Pair Simplification problem under the discrete Fréchet distance (CPS-3F). We demonstrate some novel applications and define many useful measures that have previously been unexplored.

One application we look at is based on efficiency versus cost in a hypothetical civil engineering application. Suppose we have several towns and villages along both sides of a river. We want to build bridges between the towns, but we can not afford to (nor would we want to) build bridges between every village. We want to make sure that no one in a village without a bridge has to walk more than some set distance to a town with a bridge to cross. We also have constraints on the maximum length of a bridge for safety and cost concerns.

If we minimize the number of bridges to build, this is equivalent to the minimum moving cost problem that we solve in Chapter 4. Suppose, however, that instead of minimizing the number of bridges, we want to minimize the material used, or rather the total length of all bridges. This problem takes the distance between villages into account. This is actually minimizing the discrete Fréchet distance between the simplified chains (but may not be the minimum) and is equivalent to dynamic time warping with simplification. Thus, the longest bridge is an upper bound for the discrete Fréchet distance. However, we can further complicate the problem by stating that building materials cost different amounts in different cities. This makes the optimization minimize the distance with respect to cost and is equivalent to weighted dynamic time warping with simplification.

Another interesting application is to minimize the total number of villages with a bridge for simplicity (fewer bridges) which is similar to minimizing the number of villages with only one bridge. If traffic is really heavy in one direction, we can also choose to find the minimum number needed for just one side of the river.

We consider the integral discrete Fréchet distance, and its CPS countepart. This problem is minimizing the area between the simplified curves, or can be thought of as assuring that we are staying as close to the river as possible with the bridges.

These problems are all variations of CPS-3F. CPS-3F is equivalent to asking if we can guarantee that neither side has more than some given number of bridges while all other constraints are met. Since this problem is suspected to be **NP**-complete, we build upon our earlier work with the moving cost approximation, and use the new measures to develop another approximation and better understand the properties of chain pair simplification under the discrete Fréchet distance.

5.2 A General Approach

We first define a general problem and dynamic programming solution that will solve our previous problem (CPS-3F$^+$ based on the moving cost) as well as some other properties of CPS-3F. Our original solution for the optimal moving cost was not as flexible and the recurrences can not be adapted to handle some of the other interesting characteristics of a CPS-3F solution that we will discuss.

First, we define a new 'generic' problem related to CPS-3F similar to the redefinition for CPS-3F$^+$.

Definition 21 The 'Generic' Chain Pair Simplification (gCPS) Problem.
Instance: Given a pair of 3D chains A and B, with lengths $O(m), O(n)$ respectively, an integer $K' > 0$, and $\delta_1, \delta_2, \delta_3 \in \mathbb{R}^+$.
Problem: Does there exist a pair of chains A', B' where the vertices are from A, B, respectively, $G \leq K'$, and $d_1(A, A') \leq \delta_1, d_2(B, B') \leq \delta_2, d_F(A', B') \leq \delta_3$?

For everything in this chapter $d_1 = d_2 = d_F$. The point of this problem is to reduce redundancy in defining new problems. Just by defining a new property that can be calculated as G, we can look at several interesting variations of CPS-3F. We must be careful though in defining G.

Similar to Section 4.3.1, we begin by defining our input and the conversion to rectangles. The rectangles are formally defined in Definitions 19 and 20 in Section 4.3.1. Also from Section 4.3.1, we use the definitions of $\mathcal{D}$ and $\mathcal{Q}$. We will make extensive use of the rectangle concept and these definitions throughout the rest of this chapter without referring back to the definition.

5.2.1 Recurrence

Given those definitions, we look at Equation 5.1. Here, r_p is the previous rectangle used and r_c is the current rectangle being considered in the recurrence relation.

This recurrence finds several optimal property values (including the moving cost). Thus, the cost at each step is a general function call $f(r_p, r_c)$, which is any cost that can be determined locally with only the previous and current rectangles. Below we list several of the properties and define the functions for each. Further, we show how they relate to each other, and that at least one property is also a 2-approximation for the optimal CPS-3F solution.

$$M[a_i, b_j, r_c] = \min_{(k,l)\in\{(i\text{-}1,j),(i,j\text{-}1),(i\text{-}1,j\text{-}1)\}} \begin{cases} M[a_k, b_l, r_c], & \text{if } (a_k, b_l) \in r_c \\ M[a_k, b_l, r_p] + f(r_p, r_c), & \forall\ r_p \in \mathcal{Q}_{k,l} \\ NULL, & \text{if } (a_i, b_j) \notin r_c \end{cases} \tag{5.1}$$

For the problem in Definition 21, Equation 5.1 returns the optimal solution because gCPS-3F has an optimal substructure. For the optimal substructure, we can refer back to Theorem 2, which has the same substructure with different data structures. This is easy to show by looking ahead to Section 5.3.1 where we give the function f to calculate the minimum moving cost via Equation 5.1. Thus, gCPS-3F with this function is equivalent to CPS-3F$^+$ in its substructure, and any problem which uses an $O(1)$ function based on the rectangles in gCPS-3F has an optimal substructure.

The worst case of gCPS-3F has mn rectangles, and thus the worst running time for the problem is $O(m^2n^2)$. Now we can also prove the following easily.

Theorem 5. *Given an $O(1)$ cost function f, the 'generic' chain pair simplification problem under the discrete Fréchet distance, is polynomially solvable for the optimal cost, i.e, gCPS-3F $\in \boldsymbol{P}$.*

Proof. Since gCPS-3F has an optimal substructure, given A, B, K', f, δ_1, δ_2, and δ_3, we can find an optimal K'' based on f from our dynamic programming algorithm (Algorithm 5.5). Then we decide by comparing whether $K' \leq K''$. □

5.2.2 Algorithm

Algorithm 5.5 CPS-COST $\rightarrow M$ is a 3D array holding the minimal value for a rectangle and a pair of vertices. F is the cost function which has a cost for each rectangle. This assumes $NULL$ is always ignored in min statements.

Input: i, j are indices of $a_i \in A, b_j \in B$. r_c is the current rectangle.
Output: The minimum F up to (a_i, b_j) using rectangle r_c.

```
1:  function CPS-COST (i, j, r_c)
2:      if i ≤ 0 or j ≤ 0 then return NULL
3:      if (a_i, b_j) ∉ r_c then
4:          M[i, j, r_c] ← NULL
5:          return NULL
6:      for each (k, l) ∈ {(i − 1, j), (i, j − 1), (i − 1, j − 1)} do
7:          for each r_p ∈ Q_{k,l} do
8:              if M[k, l, r_p] ≠ NULL then
9:                  M[i, j, r_c] ← min(M[i, j, r_c], M[k, l, r_p] + F(r_p, r_c))
10:             else
11:                 M[i, j, r_c] ← min(M[i, j, r_c], CPS-COST(k, l, r_p) + F(r_p, r_c))
12:     return M[i, j, r_c]
```

The dynamic programming solution based on the recurrence is straightforward and shown in Algorithm 5.5. We denote the function call as $F(r_p, r_c)$, and the individual property is easily substituted. M is a 3D array of size $|A| \times |B| \times |\mathcal{D}|$ and is assumed to be initialized to $NULL$ in every cell. The algorithm requires that min will never

choose a non-numeric value. Thus, the statement $\min(NULL, 1)$ will return 1. The algorithm also assumes that A,B, and M are globally accessible.

5.3 Cost Models

5.3.1 Moving Cost

The moving cost was defined in Definition 11 in Section 2.6. We also know from Corollary 1 that the moving cost, via CPS-3F$^+$, can be used to get a factor-2 approximation to CPS-3F. Although we defined explicit recurrences to calculate the minimum moving cost in Chapter 4, our new recurrence relation in Equation 5.1 can also calculate the optimal value by using a unit cost function for any rectangle. We also declare our problem gCPS-3F by letting G be the moving cost and then our function f is

$$f(r_{k,l}, r_{i,j}) = \begin{cases} 0, & \text{if } k = i \text{ and } l = j \\ 1, & \text{if } k \neq i \text{ or } l \neq j. \end{cases}$$

Using any new rectangle simply counts the new rectangle as 1, and if we use the same rectangle there is no cost.

As mentioned, this version of the problem only varies in the data structures used to calculate the moving cost. This difference is important though. Here, we explicitly check every possible rectangle at every step. For CPS-3F$^+$ in Section 4.3.1, we did not check rectangles after they were not optimal for the moving cost. This reduces the complexity, but also the power. The other property cost functions that follow can not be calculated using the approach of CPS-3F$^+$ in Section 4.3.

5.3.2 Number of Walks

The number of walks is the number of times that both the dog and man 'hop' at the same time (including the starting nodes). This is how many disjoint walks (Definitions 5 and 6) exist between the simplified chains. This measure is of interest because it directly affects how close the moving cost approximates the number of vertices in longer chains. We usually denote the number of walks by t, and we know the moving cost is always less than or equal to $|A'| + |B'| - t$. Thus, knowing the minimum t helps give a bound on how tight the approximation may be. We also know that $t \leq t' \leq K \leq |A'| + |B'| \leq 2K$ where t is the minimum number of walks and t' is the number of walks in the optimal CPS-3F solution yielding A', B'.

$$f(r_{k,l}, r_{i,j}) = \begin{cases} 0, & \text{if for } r_{k,l},\ k = i \text{ or } l = j \\ 1, & \text{if for } r_{k,l},\ k \neq i \text{ and } l \neq j \end{cases}$$

For gCPS-3F, G is defined as the minimum number of t-walks from Definition 5 in a paired walk between A' and B' given the other gCPS-3F constraints. The figures used in Section 2.6 are good examples of this property with its relation to the moving cost. Figure 2.3(a) has a 2-walk while 2.3(b) has a 4-walk.

5.3.3 Number of Vertices

$$f(r_{k,l}, r_{i,j}) = \begin{cases} 1, & \text{if for } r_{k,l},\ k = i \text{ or } l = j \\ 2, & \text{if for } r_{k,l},\ k \neq i \text{ and } l \neq j \end{cases}$$

An interesting method of simplification is to find the solution with the minimum number of vertices between both chains, i.e., $\min(|A'|+|B'|)$. For our general problem,

gCPS-3F, we see that $G = \min(|A'| + |B'|)$. This is another 2-approximation to CPS-3F, although not as tight as the moving cost. Also of interest is that it will give the same reduced chains A', B' as the number of walks cost model.

Corollary 2. gCPS-3F with $G = \min(|A'| + |B'|)$ gives a factor-2 approximation to the chain pair simplification problem under the discrete Fréchet distance (CPS-3F).

Proof. Given two polygonal chains A, B, let K be an optimal solution from CPS-3F yielding the simplified chains A', B', and assume K' is an optimal solution for $\min(|A'| + |B'|)$ yielding the simplified chains A'', B'', i.e. $K' = \min(|A''| + |B''|)$.

Here, $K \leq K'$ because K is the minimum number of vertices of $|A''|+|B''|$ possible which is at least equal to $\max(|A'|, |B'|)$. Now, $K' = |A''| + |B''| \leq |A'| + |B'| \leq 2\max(|A'|, |B'|) = 2K$. Thus, $K \leq K' \leq 2K$. □

It is also of interest that this version correctly solves the counter-example shown in Section 4.3.3. However, we can construct a counter-example for this problem as well. It is easy to imagine cases where it occurs that the minimum number of vertices does not result in the minimum K of CPS-3F. Suppose $|A'| = |B'| = 10$ and thus $K = 10$. Now suppose there is another possible solution such that $|A'| = 11$, $|B'| = 8$, and thus $K = 11$. The sum where $K = 11$ is 19, yet the minimum K simplification yields a sum of 20. Although no explicit one is listed here, it becomes trivial to show they exist in larger chains of several hundred vertices. Cases where $|A'| = |B'| = 500$ is one solution and another is $|A'| = 501$, $|B'| = 498$ are certainly possible.

5.3.4 Minimum Sum

Calculating the minimum total distance across the simplified chains is equivalent to an optimal dynamic time warping solution with simplification. The cost function

simply takes the distance of the nodes used into account. Note that gCPS-3F based on the minimum sum is equivalent to standard DTW if we let $\delta_1 = \delta_2 = 0$ and $\delta_3 = \infty$.

$$f(r_{k,l}, r_{i,j}) = d(a_i, b_j)$$

This is of interest because almost no work has been done on dual simplification of the chains for an optimal dynamic time warping solution. The complexity of dynamic time warping without simplification is $O(mn)$, and this shows that the problem under simplification is at most $O(m^2n^2)$. Thus, we can find optimal solutions for CPS under dynamic time warping, and this is still preferable over RMSD. RMSD must be pairwise compared with each chain having the same number of vertices. This makes some comparisons impossible without ignoring parts of the chains, but this is not a problem with DTW, and thus dynamic time warping may be a more natural fit for applications like protein alignment, comparison, and simplification.

There are many techniques used in the dynamic time warping literature to reduce the complexity or running time for large chains by pruning the recursive search space. These strategies might also be effective to keep CPS-3F$^+$ (Section 4.3) running in near $O(mn)$ time. The most commonly used DTW pruning techniques are the Itakura Parallelogram [38] and the Sakoe-Chiba Band [39].

5.3.5 Integral CPS

There is no known algorithm for the integral Fréchet distance [40], and we have not pursued any relating to the discrete Fréchet distance, but we can find the minimum integral value based on chain pair simplification. Integral chain pair simplification is related to the minimum sum version, but can differ greatly in higher dimensions or

with other distance metrics. This function finds the minimum two-dimensional area between the two chains by minimizing the area of the 2D polygons between the two chains.

The calculation for $f(r_{k,l}, r_{i,j})$ is related to the number of walks. For a new walk, the area is rectangular, but if the new pair of nodes is part of an existing walk, then the calculation is a triangle. The equation below shows this simply and abstracts the actual calculation of the area with a generic $AREA()$ function for clarity.

$$f(r_{k,l}, r_{i,j}) = \begin{cases} AREA(a_i, a_k, b_j), & \text{if } i \neq k \text{ and } j = l \\ AREA(a_i, b_l, b_j), & \text{if } i = k \text{ and } j \neq l \\ AREA(a_k, b_l, b_j, a_i), & \text{if } k \neq i \text{ and } l \neq j \end{cases}$$

For our applications, the integral-based simplification was unnecessary, but is an interesting use of chain pair simplification that may prove useful in other applications.

5.4 Optimal One-sided Simplification

Although the complexity of CPS-3F is unknown, it is possible to find the minimum size of one of the chains with respect to the simplification if we are unconcerned with the size of the other chain. Thus, given a CPS-3F instance, we can find an optimal solution for $|A'|$ or $|B'|$, which will be independent of the solution for the other chain.

$$M[i, r_c] = \min \begin{cases} M[i-1, r_c], & \text{if } x_{i-1} \in r_c \\ M[i-1, r_p] + f(r_p, r_c), & \forall r_p \; s.t. \; x_{i-1} \in r_p, (\mathcal{Q}_{i-1}) \\ NULL, & \text{if } x_i \notin r_c \end{cases} \tag{5.2}$$

This gives us a bound on K. There are two scenarios. The first is if our two simplified chains when minimizing $|A'|$ are A'_A, B'_A and $|B'_A| \leq |A'_A|$, then $K = |A'_A|$ as the solution for CPS-3F (and similarly with A'_B, B'_B being the chains when finding a minimal $|B'|$). The more likely case is that $|A'_A| < |B'_A|$ in which we only know that $|A'_A| \leq K \leq |B'_A|$.

Assume we have solved for the optimal solution for each chain where the simplified chains for A are A'_A, B'_A, and the minimum solution for chain B yields A'_B, B'_B such that $|A'_A| < |B'_A|$ and $|B'_B| < |A'_B|$. Then $\max(|A'_A|, |B'_B|) \leq K \leq \min(|A'_B|, |B'_A|)$.

None of these properties work for an approximation solution. It is possible for $|A'_A| + |B'_B| < K$ and also possible for $2K < \min(|A'_B|, |B'_A|)$. We do know that in relation to the minimum number of both (Section 5.3.3) the inequality $\min(|A'_A|) + \min(|B'_B|) \leq \min(|A'| + |B'|)$ holds. Regardless, the bound guarantee is still useful and worth the exploration.

Equation 5.2 again uses a generic function f for the cost. All of the cost models from Section 5.3 can be used, but only the optimal solution with respect to one of the chains will be calculated, e.g., the minimum number of vertices is the minimum number of vertices for one chain. Similar to our notation in the previous chapter, $\mathcal{Q}_i$ is the set of rectangles that cover x_i when X is the chain that we are finding an optimal solution for. We give the function for the minimum number of vertices below, but do not investigate the other cost models directly due to space and redundancy.

$$f(r_p, r_c) = \begin{cases} 0, & \text{if } p = c \\ 1, & \text{if } p \neq c \end{cases}$$

Algorithm 5.6 implements Equation 5.2 assuming a generic X as the target chain to simplify, and that the calculation of rectangles between X and the other chain has

Algorithm 5.6 OPT-SINGLE-CPS $\rightarrow M$ is a 2D array holding the minimal value for a rectangle and a pair of vertices. F is the cost function which has a cost for each rectangle. This assumes $NULL$ is always ignored in min statements.

Input: i is the index of the chain to optimally simplify. r_c is the current rectangle.
Output: The minimum F up to x_i in chain X using rectangle r_c.

```
1: function OPT-SINGLE-CPS (i, r_c)
2:     if i ≤ 0 then return NULL
3:     if x_i ∉ r_c then
4:         M[i, r_c] ← NULL
5:         return NULL
6:     for each r_p ∈ Q_{i−1} do
7:         if M[i − 1, r_p] ≠ NULL then
8:             M[i, r_c] ← min(M[i − 1, r_c], M[i − 1, r_p] + F(r_p, r_c))
9:         else
10:            M[i, r_c] ← min(M[i, r_c], OPT-SINGLE-CPS(i − 1, r_p) + F(r_p, r_c))
11:    return M[i, r_c]
```

already been done and stored in $\mathcal{Q}$. The algorithm also assumes the two chains and M are globally accessible. M is a 2D array of size $|X| \times |\mathcal{D}|$ where X is either A or B, and M is assumed to be initialized to $NULL$ in every cell. Similar to Algorithm 5.5, the algorithm requires that min will never choose a non-numeric value. Thus, the statement $\min(NULL, 1)$ will return 1.

The complexity has a worse case of $O(|X|^2|Y|)$ between two polygonal chains X and Y and we are finding the optimal size of chain X. This is because there may be $|X| \cdot |Y|$ rectangles.

5.5 Directed Acyclic Graph Approach

We have largely approached this problem in a way that analyzes the matrix of possibilities between pairs of nodes between the two chains. Now, we look at converting an instance of CPS-3F into a weighted directed acyclic graph (DAG). This allows another level of abstraction by using the edges to relate overlapping and adjacent

rectangles. This also allows a lot of simplification for finding solutions to many of the related problems we have discussed (such as CPS-3F$^+$). For reference, Figure 5.1 shows a simple example of the difference. This solution is using the example from Section 4.3.3. We can see the two paths made from the rectangles and the resulting DAG overlayed. The difference in rectangle color is merely for clarity with overlap.

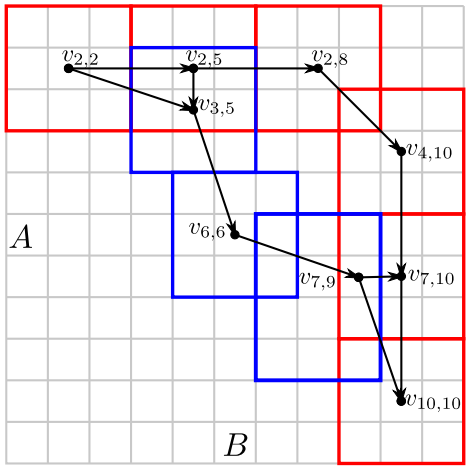


Figure 5.1: The CPS-3F rectangles built for the chains in Table 4.1 showing two possible solution paths. A DAG built from these rectangles is overlayed along with the vertex numbers, which are equivalent to the rectangle numbers.

5.5.1 Define Rectangles

Given two polygonal chains $A = \langle a_1, a_2, ..., a_m \rangle$, and $B = \langle b_1, b_2, ..., b_n \rangle$, and constraints $\delta_1, \delta_2, \delta_3 \in \mathbb{R}^+$, and a $K \in \mathbb{Z}^+$. First, let $\mathcal{D} = \{(a_i, b_j) |\ a_i \in A,\ b_j \in B$ and $d(a_i, b_j) \leq \delta_3\}$ as in Section 4.3.1. This is the set of all pairs of nodes between the two chains that are at a distance of at most δ_3 from each other. We also use rectangles as specified in Definitions 19 and 20 in Section 4.3.1. For simplicity and clarity in this section we will use r_{ij} instead of $r_{i,j}$ for a rectangle based on the pair (a_i, b_j) where $a_i \in A$ and $b_j \in B$.

Lemma 3. Any solution to CPS-3F can be represented as a sequence of adjacent rectangles $R = \langle r_1, r_2, \ldots, r_p \rangle$ where each r_i is defined as in Definition 20.

Proof. We know that for $d_F(A', B') \leq \delta_3$, any solution to CPS-3F will only use pairs of vertices between A, B that are also within δ_3, i.e., pairs in the set $\mathcal{D}$. Given every $(a_i, b_j) \in \mathcal{D}$ is represented by the rectangle r_{ij}, the only issue is the maximal rectangles themselves. Assume there was an optimal solution S which is the sequence of pairs from $\mathcal{D}$ that make up A', B', but that can not be represented by the rectangles R based on those pairs in S. Since $d_F(A', B') \leq \delta_3$ via the pairs in S and their analogues in R, the problem must be between A, A' or B, B'. If we assume that the solution S maintains $d_F(A, A') \leq \delta_1$ and $d_F(B, B') \leq \delta_2$, then we have reached a contradiction. Since for a pair $(a_i, b_j) \in S$, r_{ij} would represent all possible simplified vertices since it is maximal. Thus, any solution can be represented as a sequence of adjacent rectangles. □

The time to do this is polynomial, and Algorithm 5.7 takes $O(mn)$ time to find $\mathcal{D}$ while Algorithm 5.8 is $O(n)$ for each chain. MARK-RECTS is straightforward. We mark the nodes with the name of the rectangle and a tick mark to differentiate the nodes in $\mathcal{D}$.

MARK-CHAIN analyzes a single chain and assumes that the nodes in $\mathcal{D}$ have already been marked. This scans in one direction marking all nodes within δ after a marked node has been seen. Note that since it is one-directional, MARK-RECTS must call it twice for each chain in each direction. We assume the language knows whether to increment or decrement based on the start S and end E values.

Algorithm 5.7 MARK-RECTS → Find all the pairs of nodes in $\mathcal{D}$ and mark those nodes with the rectangle name.

Input: Polygonal chains A and B. Distance parameters δ_1, δ_2, and δ_3.

```
1:  procedure MARK-RECTS (A, B, δ1, δ2, δ3)
2:      //Find nodes in D and mark with rectangle and tick
3:      for each a_i ∈ A do
4:          for each b_j ∈ B do
5:              if d(a_i, b_j) ≤ δ3 then
6:                  MARK(a_i, r'_ij),MARK(b_j, r'_ij)
7:      //Mark nodes in A with rectangles
8:      MARK-CHAIN(A, δ1, 1, |A|)
9:      MARK-CHAIN(A, δ1, |A|, 1)
10:     //Mark nodes in B with rectangles
11:     MARK-CHAIN(B, δ2, 1, |B|)
12:     MARK-CHAIN(B, δ2, |B|, 1)
```

Algorithm 5.8 MARK-CHAIN → Mark each node of both chains with all rectangles that they are in.

Input: Polygonal chain X. Distance parameter δ. The indices of nodes within X to start from (S) and end at (E).

```
1:  procedure MARK-CHAIN (X, δ, S, E)
2:      for i ← S to E do
3:          //If x_i has been marked by a rectangle with a tick
4:          if MARKED(x_i) then
5:              //Add all ticked rectangles as unticked to marked-rects
6:              ADD(marked-rects,GET-MARKED-RECTS(x_i))
7:          //Go through current list of marked-rects
8:          for each rect ∈ marked-rects do
9:              // rect_x = x_i for A and x_j for B
10:             if dist(rect_x, x_i) ≤ δ then
11:                 MARK(x_i, rect)
12:             else
13:                 REMOVE(marked-rects,rect)
```

5.5.2 Creating a WDAG

We now create a weighted directed acyclic graph (WDAG) $G = \{V, E\}$ as follows. Convert each rectangle r_{ij} into a vertex v_{ij}, which means $|V| = |\mathcal{D}|$. We

create a directed edge from v_{ij} to v_{kl} if $\max(S_A(a_i, \delta_1)) \geq \min(S_A(a_k, \delta_1)) - 1$ or $\max(S_B(b_j, \delta_2)) \geq \min(S_B(b_l, \delta_2)) - 1$ and $i \leq k$, $j \leq l$.

Definition 22. The inset of v_{ij}, $N^-(v_{ij})$, is all r_{kl} touching r_{ij} s.t. $k \leq i$, $l \leq j$ and the outset, $N^+(v_{ij})$, is all touching r_{kl} s.t. $k \geq i$, $l \geq j$.

We can ignore rectangles that touch, but do not respect the subscript constraint. Given the discrete Fréchet distance is a monotonic function, we can not have a vertex v_{ij} pointing to a vertex v_{kl} where $i > k$ or $j > l$. This would imply the weak discrete Fréchet distance because it is not a monotone function.

We label all vertices that cover (a_1, b_1) as start nodes and they can not have edges pointing to each other (although this condition is not necessary). Any node that covers (a_m, b_n) is marked as an end node. Similarly, an end node should not point to other nodes. Finally, our stopping condition sets K to the minimum K_{ij} of all end nodes.

Lemma 4. We can create a weighted directed acyclic graph from a CPS-3F instance such that any path from a start vertex to an end vertex means every vertex is covered.

Proof. As we defined the directed edges, they only exist between nodes if the min and max S_X chains cross or are immediately adjacent for both A and B. Thus, if a path exists from a start node, which covers (a_1, b_1), to an end node, covering (a_m, b_n), then all nodes of A and B are covered. The graph is also guaranteed to be acyclic via Lemma 22. □

Corollary 3. Any CPS-3F solution will exist as a path in our DAG from a start vertex to an end vertex.

Proof. This is equivalent to Lemma 3 as a graph. □

5.5.3 Minimum K_{ij} for a Vertex

For each vertex v_{ij}, let C_{ij} be a set of pairs $C_{ij} = \{(c_{a_1}, c_{b_1}), (c_{a_2}, c_{b_2}), \ldots, (c_{a_F}, c_{b_F})\}$ and suppose F is a bounded constant (we prove this in the next section). Each pair represents one possible path through the graph and (c_{a_p}, c_{b_p}) are the lengths of A', B', respectively, up to vertex v_{ij}. We will call these "path sets". The weight for any edge, $w(e)$, is in the set $\{(0,1),(1,0),(1,1)\}$, and we will subscript them as $w_a(e), w_b(e)$. Given an edge $e_{ij,kl}$ from v_{ij} to v_{kl} the weight is

$$w(e_{ij,kl}) = \begin{cases} (0,1) & \text{if } i = k, j < l \\ (1,0) & \text{if } i < k, j = l \\ (1,1) & \text{if } i < k, j < l. \end{cases}$$

This means we are increasing the size of the path in either A', B', or both. We can then calculate the minimum K_{ij} for a vertex v_{ij} based on its inset by

$$K_{ij} = \min_{\forall (c_{a_p}, c_{b_p}) \in C_{kl} \text{ of } v_{kl} \in N^-(v_{ij})} \{\max(c_{a_p} + w_a(e_{ij,kl}), c_{b_p} + w_b(e_{ij,kl}))\}.$$

All nodes processed will have an inset unless they are a start node. If a vertex does not have an inset, and it is not a start node, then it is disconnected from the graph and we can ignore it.

Start nodes have a path set $C_s = \{(1,1)\}$ and for an end node, v_{ij}, the min K path is $K = \min K_{ij}$.

Lemma 5. If F is bounded by a constant, then K_{ij} can be processed in time $O(QF)$ time where $Q = |N^-(v_{ij})|$.

Proof. This is straightforward. In a complete graph the maximum number of edges to any node is n^2 where $n = |V|$. Thus, multiplying n^2 vertices by a constant Q for all n^2 edges is $O(Qn^4)$. Our bound is much lower since we have constraints on what vertices can be connected and can be directed to a vertex. □

5.5.4 Bounded Path Sets

Since we know that if F is a constant, we can calculate the optimal path at a vertex, we now prove that $F = K - K_{ij} + 1 \leq 2(K - K_{ij} + 1) \leq 2K$.

Lemma 6. For any vertex v_{ij}, the path set C_{ij} is of size at most F where $F \leq 2(K - K_{ij} + 1) \leq 2K$.

Proof. Let $(c_a, c_b) \in C_{ij}$ such that $c_a \geq c_b$ and $K_{ij} \leq c_a \leq K$ by definition. If we fix c_a to a specific value T in this range, then there are three cases:

(1) Suppose there is another path length pair where $(c_{a_p}, c_{b_p}) = (c_a, c_b)$. We can ignore this path because we already know we can get to v_{ij} with lengths of A', B' as c_a, c_b, respectively.

(2) Suppose there is a path length pair (c_{a_p}, c_{b_p}) such that $c_{a_p} = c_a$ and $c_{b_p} < c_b$. In this case the chain has $c_{b_p} < c_b \leq K_{ij}$. If this is true, then there exists a path to v_{ij} where B' is smaller. Since the path taken is unimportant, set $(c_a, c_b) = (c_{a_p}, c_{b_p})$. Now we have the smallest path length of both chains A', B' to v_{ij} where $c_a = T$.

(3) Suppose there is a path length pair (c_{a_p}, c_{b_p}) such that $c_{a_p} = c_a$ and $c_{b_p} \geq c_b$. Since any subsequent path will have the same edge weights, (c_{a_p}, c_{b_p}) will always have lengths greater than or equal to (c_a, c_b), so we can ignore this path.

Thus, for any fixed $c_a = T$ where $c_a \geq c_b$ and $K_{ij} \leq c_a \leq K$ we only need to save one cost pair in C_{ij}. We must do this for each value of c_a between K_{ij} and K, which is $K - K_{ij} + 1$ values. This is identical for cases where $c_b \geq c_a$. Thus, there are

$2(K - K_{ij} + 1)$ total pairs. With $C_{ij} = \{(c_{a_1}, c_{b_1}), (c_{a_2}, c_{b_2}), \ldots, (c_{a_F}, c_{b_F})\}$, we know $F \leq 2(K - K_{ij} + 1) \leq 2K$. □

5.5.5 Summary

Even though each vertex needs to store less that F possible paths, for a vertex v_{ij} the recurrence still has to look at $i \cdot j - 1$ possible connected nodes. Thus, naïvely running the recurrences on this graph for an optimal CPS-3F solution would be exponential in complexity. For a worst case, there would be mn vertices and m^2n^2 edges. Thus, although we have proven a bound on what a single node would need to store, the number of possible paths is still exponential.

The graph does allow us to simplify our other approximation algorithms and only deal with the nodes that are part of the simplified chains (pairs in $\mathcal{D}$). Given the graph is already created, it should also decrease the running time of the dynamic programming solutions. However, as mentioned the worst case still has mn vertices and m^2n^2 edges, so similar to our other solutions, the complexity could be as bad as $O(m^2n^2)$.

CHAPTER 6

DISCRETE SET-CHAIN MATCHING

In this chapter we examine the discrete version of a problem originally defined in [12] with the continuous Fréchet distance, which we will call the set-chain matching problem. Following, we define and analyze other variations of this problem that are also of interest. Figure 6.1 shows a simple instance of the problem where we have a set of points and a polygonal curve, and we wish to find another polygonal curve through the set of points such that the Fréchet distance between the two curves is below a given threshold.

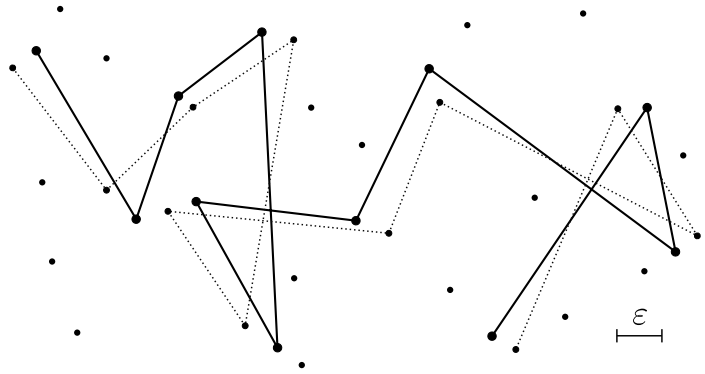


Figure 6.1: An instance of the discrete set-chain matching problem in 2D with one possible solution of $|Q| \geq 11$ for the shown ε.

The set-chain matching problem is equivalent to the map matching problem for complete graphs. Essentially, there is more freedom because any choice of points is possible. In a graph, however, you are restricted by the neighbors at any given vertex. We first formally define the problem and its variations.

6.1 Application

The discrete set-chain problem has many applications depending on which variation is being considered. Suppose we have intermittent lossy GPS vehicle data where we can not guarantee the path of the vehicle between our data points. We can find the shortest (and arguably the most plausible) path of the vehicle.

If the points in our set represent signal towers (cellular, radio, etc.), which generally have a spherical range, then we can also consider several coverage problems. Assuming we know the path of a vehicle, what is the minimum number of towers needed to ensure that the signal is not lost. Simply knowing whether the path is covered is important, but optimizing it along multiple roads and areas is crucial. These types of problems are studied in many areas including wireless sensor networks, graphics, scheduling, and ordering.

6.2 Related Work

The set-chain problem with the continuous Fréchet distance was defined by Maheshwari et al. as trying to find a polygonal curve through a subset of points to minimize the Fréchet distance to some curve P. We call this problem set-chain matching to avoid confusion with other types of matching problems. Although they do not specify the name for the problem, they define the set-chain matching problem as: Given a point set S and a polygonal curve P in $\mathbb{R}^d$ ($d \geq 2$), find a polygonal curve Q with its vertices chosen from S, which has a minimum Fréchet distance to P.

For our investigation, we use the discrete Fréchet distance and thus call it the discrete set-chain matching problem. As we will show, a variation of the discrete

set-chain problem is also related to the discrete unit disk cover (DUDC) problem when minimizing the number of points from S used. The DUDC problem is known to be **NP**-Hard, and is also difficult to approximate with the most recent results being an 18-approximation algorithm [41], a 15-approximation algorithm [42], and a $(9+\varepsilon)$-approximation algorithm [43]. Nearly all of the constant factor approximations have been within the last decade. The problem does admit a PTAS [44], but this is infeasible for most instances of the problem. The problem does not admit a Fully Polynomial Time Approximation Scheme (FPTAS) unless **P**=**NP**.

6.3 Discrete Set-chain Matching

We begin with the formal definitions of the problem and the variations as well as some terminology. It is important to note that, as in the continuous version, we make no requirements that P or Q be planar. Further, for problem classification we mainly focus on the reachable points (defined below). For discussion, we will refer to the number of nodes in a polygonal chain as the "size" of the chain and it will be denoted as $|A|$ for a polygonal chain A.

Definition 23 The Discrete Set-Chain Matching Problem.
Instance: Given a point set S, a polygonal curve P in $\mathbb{R}^d$ ($d \geq 2$), an integer $K \in \mathbb{Z}^+$, and an $\varepsilon > 0$.
Problem: Does there exist a polygonal curve Q with vertices chosen from S' where $S' \subseteq S$, such that $T \leq K$ and $d_F(P, Q) \leq \varepsilon$?

T is defined in two ways. When we are minimizing the number of nodes in the chain, $T = |Q|$, and if we are minimizing the points used then $T = |S'|$. Figure 6.2 shows an example case demonstrating the difference between minimizing $|Q|$ or $|S'|$.

Here, minimizing $|Q|$ will always yield $|Q| = 3$ regardless of the points chosen as nodes. However, minimizing $|S'|$ will return $|S'| = 2$ and $|Q| = 3$ and there is only one possible sequence of points that is minimal.

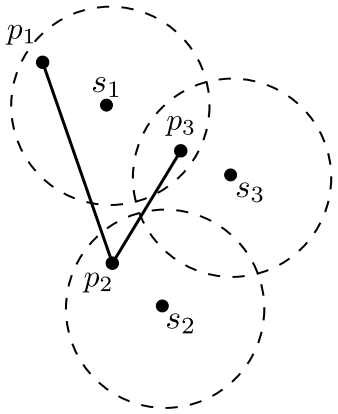


Figure 6.2: The difference between minimizing $|Q|$ and $|S'|$. Minimizing $|S'|$ gives $Q = \langle s_1, s_2, s_1 \rangle$ where $|S'| = 2$ and $|Q| = 3$, but minimizing $|Q|$ will yield $|Q| = 3$ whether it uses the sequence $\langle s_1, s_2, s_1 \rangle$ or $\langle s_1, s_2, s_3 \rangle$.

Now define $S_\varepsilon = \{s \in S | p \in P \text{ and } d(p, s) \leq \varepsilon\}$ as the reachable points because these are the points s that can be reached by p within the ε given. Here, we look at four variations of this problem. They vary whether or not there is a uniqueness constraint on $s \in S$ being used as a node in Q (if points may be used more than once), and whether our goal is to minimize the size of the chain Q or of the set S'. We therefore distinguish the problems as Unique/Non-unique(U/N) Set-Chain(S) Matching(M) with a k Subset/Chain(S/C). The variants are thus NSMS-k, NSMC-k, and USM-k. When looking at unique nodes, minimizing $|Q|$ is equivalent to minimizing the set of points used, $|S'|$, since they can only be used once, so we do not separate the cases.

6.4 NSMC-k

We begin by first showing that NSMC-k exhibits an optimal substructure. Figure 6.2 points out that we merely must find at least one point $s_i \in S$ for every $p_j \in P$. The only exception is to find an s that covers consecutive p's. This can be done by looking at the neighbors of a node, and selecting the points that cover them. The recurrence relation is shown in Equation 6.1. We declare a 2D array of size $|S| \times |P|$ where the columns represent the nodes in the polygonal chain P and the rows represent points in the set S. For our initial condition, we assume a column 0 that is populated with 0's in every row. Now the recurrence can be processed column by column until finished. The final optimal value will be $Opt = \min_{k=1}^{|S|}(M[k, |P|])$. This can be solved in $O(mn)$ time. We show a simple iterative algorithm that implements this method in Algorithm 6.9.

$$M[i,j] = \min \begin{cases} M[i, j-1], & \text{if } d(s_i, p_j) \leq \varepsilon, M[i, j-1] \neq \emptyset \\ \min_{k=1}^{|S|}(M[k, j-1]) + 1, & \text{if } d(s_i, p_j) \leq \varepsilon, M[i, j-1] = \emptyset \\ \emptyset, & \text{if } d(s_i, p_j) > \varepsilon \end{cases} \tag{6.1}$$

Theorem 6 (Optimal Substructure of NSMC-k)**.** *Let $P = \langle p_1, ..., p_n \rangle$ be a polygonal chain, and $S = \{s_1, ..., s_m\}$ be a set of points such that there exists a $Q = \langle q_1, ..., q_k \rangle$ through a set $S' \subseteq S$ which is a minimal sequence such that $d_F(P, Q) \leq \varepsilon$.*

(1) If $d(p_{n-1}, q_k) \leq \varepsilon$ and $d(p_{n-1}, q_{k-1}) > \varepsilon$, then Q_k is an optimal solution for P_{n-1}.

(2) If $d(p_{n-1}, q_{k-1}) \leq \varepsilon$, then Q_{k-1} is an optimal solution for P_{n-1}.

(3) If $d(p_{n-1}, q_k) > \varepsilon$, then Q_{k-1} is an optimal solution for P_{n-1}.

Proof. (1) If $d(p_n, q_k) \leq \varepsilon$ and $d(p_{n-1}, q_k) \leq \varepsilon$, then the point q_k covers both points by an ε-ball. However, q_{k-1} does not cover p_{n-1}. Thus, Q_k is still the optimal solution. (2) If $d(p_{n-1}, q_{k-1}) \leq \varepsilon$, then q_k only covers p_n. If $d(p_n, q_{k-1}) \leq \varepsilon$, then Q_{k-1} would be an optimal solution, but by definition Q was minimal so this can not be true. (3) If $d(p_{n-1}, q_k) > \varepsilon$, then we have the same argument with p_n only covered by q_k, and thus Q_{k-1} must be optimal for P_{n-1}. □

Theorem 7. *The discrete non-unique set-chain matching problem where $T = |Q|$ is polynomial, i.e., NSMC-k $\in$ **P**.*

Proof. Since we have shown that NSMC-k has an optimal substructure, given P, S, and K, we can find an optimal K' from a dynamic programming algorithm based on the recurrences (Equation 6.1). Then we decide NSMC-k by comparing whether $K \leq K'$. □

6.5 NSMS-k

We now look at the discrete non-unique set-chain matching problem where we want to minimize the number of points from S used in Q. This is now a coverage issue, and the problem becomes extremely difficult. This problem is equivalent to the discrete unit disk cover (DUDC) problem, which is known to be **NP**-Hard and is difficult to approximate.

Theorem 8. *The discrete non-unique set-chain matching (NSMS-k) problem where $T = |S'|$ is **NP**-complete.*

Proof. This can be shown via a straightforward reduction from the discrete unit disk cover (DUDC) problem which is **NP**-Hard [41]. Formally, we are given a set of points

Algorithm 6.9 OPTIMAL-NSMC → Return the minimal size of $|Q|$.

Input: P is the polygonal curve, S is the set of points, ε is our distance threshold, and k is our optimization limit.

Output: The minimum $|Q|$ possible for a Q through the points in S such that $d_F(P, Q) \leq \varepsilon$.

```
1: function OPTIMAL-NSMC (P, S, ε, k)
2:     M ← |S| × |P|
3:     LastMin ← 0
4:     for j ← 1, |P| do
5:         CurrentMin ← LastMin +1
6:         for i ← 1, |S| do
7:             if d(s_i, p_j) ≤ ε then
8:                 if M[i, j − 1] = NULL or j = 1 then
9:                     M[i, j] ← LastMin + 1
10:                else
11:                    M[i, j] ← M[i, j − 1]
12:            else
13:                M[i, j] ← NULL
14:            if M[i, j] < CurrentMin then
15:                CurrentMin ← M[i, j]
16:    return CurrentMin
```

P and a set of disks $D = \{D_1, D_2, ..., D_N\}$ with centers $C = \{c_1, c_2, ..., c_N\}$ with all disks of radius r.

Now, let P' be a polygonal chain made of all points in P in any order. Let $S = C$ and $\varepsilon = r$. Now, $\exists$ a minimum-cardinality subset $D' \subseteq D$ with centers C' such that $\forall\ p \in P$, $\exists$ a $D_i \in D'$ that contains p if and only if $\exists$ a polygonal chain Q where the vertices are from points in $S' \subseteq S$ such that $|S'| = |D'|$ and $d_F(P', Q) \leq \varepsilon$.

We first prove the forward direction. Given an instance $I \subseteq D$ that is a minimum covering for all points in P. We construct P' by connecting all points in P in any order. Making a polygonal chain Q with the set of centers (C_I) of I is straightforward. We construct Q by finding the disk (D_i) that covers $p_1 \in P'$, and we set $q_1 = c_i$ where c_i is the center of disk D_i. Similarly, we walk through each $p_i \in P'$ and set the center

of the disk $D_j \in I$ covering point p_i as $q_i = c_j$. Every ordered node in P' is now still within ε of a node in Q, thus $d_F(P', Q) \leq \varepsilon$, and the set of nodes used, $|S'|$, is equal to $|I|$.

In the other direction, if we have a polygonal chain $Q = \{q_1, q_2, \ldots, q_N\}$ such that the number of unique locations used for vertices is of minimal cardinality and $d_F(P', Q) \leq \varepsilon$. Suppose the set of unique locations S' that Q is made of is not a minimal disk cover of all the vertices of P' viewed as points in a set P. This implies there exists at least one q_i that is unnecessary for a covering by C, and there is a point p_j that can be covered by another c_k. Let C' be this smaller covering. Using the same construction as above we can build a P'' and Q'. This would mean $|C'| < |S'|$ which contradicts our assumption that S' in minimal. Thus, every node $p_i \in P'$ is within ε of at least one node $q_j \in Q$, and S' is a minimal cover.

Finally, we show the problem is in **NP**. Given an instance I we can check whether $d_F(P, I) \leq \varepsilon$ in $O(mn)$ time via Theorem 1. □

6.6 USM-k

This version of the problem is shown to be **NP**-complete via a reduction from planar 3-SAT [45]. Again, we set $k = |S_\varepsilon|$ requiring all reachable points to be used. By standard convention, we first introduce several planar "gadgets" that we then arrange in our reduction. We will build up the gadgets in a piecewise manner, and then show how they are connected to form a single polygonal line at the end.

The first gadget we look at is the choice gadget. This is the building block for the reduction. Here, in Figure 6.3(a), we can see that in order for a line through the points to 'cover' the nodes of the line, there are two ways for the new line to go. We

label the one that goes to the peak first, and then the point in the middle as 'true', and if it goes the the middle one first, then the peak it is 'false'.

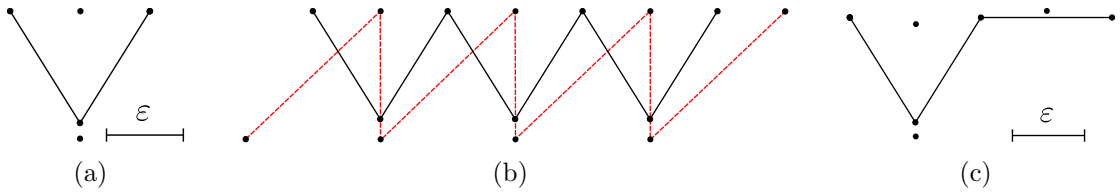


Figure 6.3: (a) A choice. (b) A chain with a false connection. (c) A variable gadget.

We can now link these choice gadgets together to make a chain. The chain is important because we can use it to force the new line to stay in the 'true' or 'false' orientation. Examples of these two paths within a chain are shown in Figures 6.3(b).

The variable gadget is similar to the chain, except it also contains an extra line and node that, depending on whether the new line is 'true' or 'false', will use or not use. Figure 6.4 shows the variable gadget for x_i and both the 'true' and 'false' settings. If the new line is 'true', then the variable does not need the extra point as a node, however, a 'false' line will use the point because the previous point was already used. As is standard in many reductions, each variable is repeated some finite length in one direction based on what is needed in the equation (only thrice in Figure 6.4).

Unfortunately, the variable gadget alone will not ensure that the new line alternates between 'true' and 'false' states, which we need for a variable and its complement. Therefore, we modify the variable gadget with a "switch" gadget, which makes the free point necessary at every other variable link, and thus alternates the path of Q. It is important to note that these switch segments will not be directly connected to the variable gadgets. These are clear in the examples in Figure 6.4, and that the first and last instance are not the full 'U' shaped gadget.

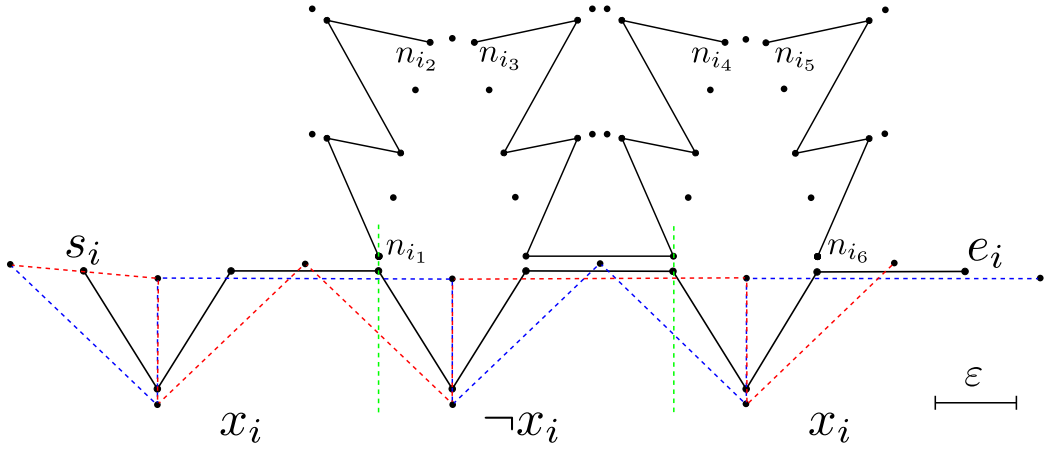
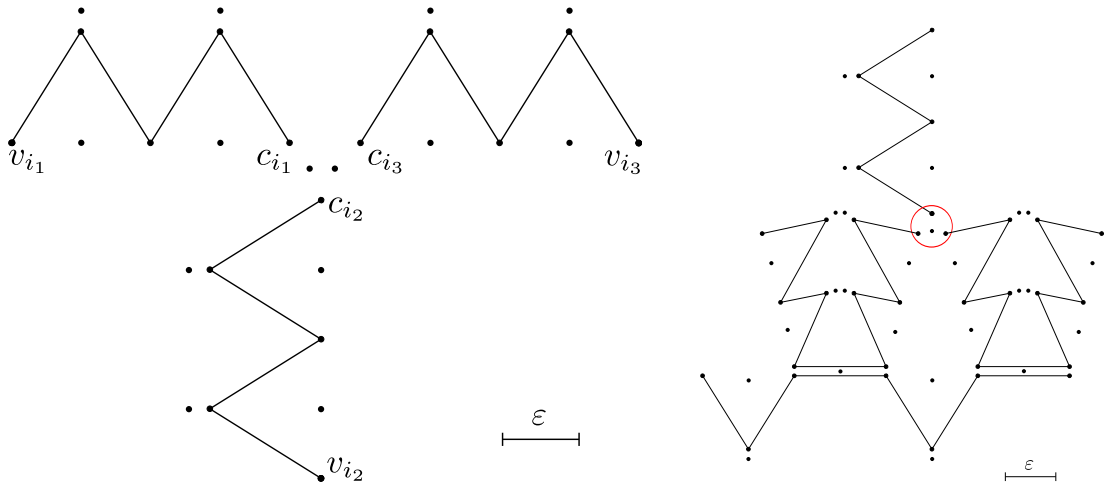


Figure 6.4: Variable gadgets linked together for variable x_i. Both 'True' and 'False' settings are shown as well as the division between x and $\neg x$.

A clause gadget is rather simple. Three chains, each connected at the other end to a variable, come together and end within ε of each other, and there are only two points between the ends. Figure 6.5(a) gives an example of this. At least one of the chains must have the new line coming through the points in a 'true' position so that its final node (C_{k_i}) at the end of the chain does not need one of the two mutual points. Simply, only two of the chains can have a 'false' setting or one of the end nodes in the clause will not be covered within ε, and the clause would be false.

The chains from the clause gadgets are attached to the variable/switch gadgets in the highlighted area of Figure 6.5(b). There is one point between the ends of the three chains. A line is added from the clause endpoint v_{k_y} (for clause c_k where $1 \leq y \leq 3$) to the opposite side of the variable (or complement) in the clause, e.g., if x_1 is the third variable in the clause c_k and the connection point is $n_{1_i}(x_1)$ or $n_{1_j}(\neg x_1)$, then a line is placed connecting the chain $v_{k_3}$to $n_{1_j}(\neg x_1)$.



(a) Clause Gadget (b) Variable Connection

Figure 6.5: (a) The clause gadget. (b) The connection for the chain to the clause.

Now we have to show that all of the ends can be connected without creating loops. The polygonal line, P, does not have to be planar, however it must be a single continuous line. The non-planarity allows us to focus on a single clause to show one way in which everything can be connected. We have to be careful that we do not connect two ends that would change the reduction such as connecting two end nodes at a clause– $c_{k_1}, c_{k_2}, c_{k_3}$ for clause C_k. For simplicity, we can connect all variables together and all the beginning and end switch points. Let $q_1 = s_1$ and then connect the variables by adding in the edge e_k, s_{k+1} for all variables $1 \leq k \leq N-1$, and the last variable node e_n connects to a vertex in c_1.

We show a simple example of three variables and a clause in Figure 6.6 without the ends being connected. If we assume the example is clause C_k, and the clauses connect to the variables,x_1, x_2, x_3,at nodes n_{p_i} or n_{p_j} where $1 \leq p \leq 3$ and n_{p_j} is the other end of the segment connected to n_{p_i}. We are only concerned about the unconnected segment ending at the clause. The other segments will be taken care of

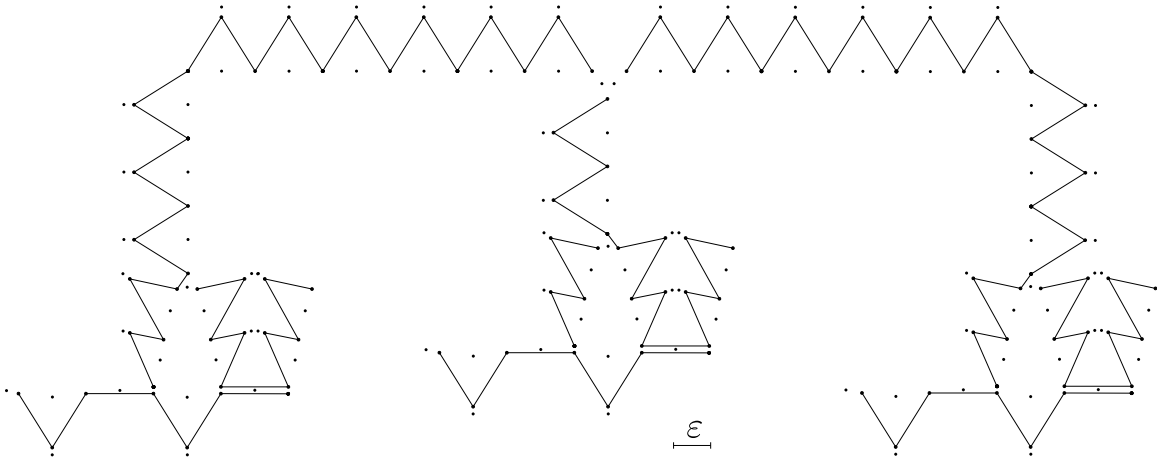


Figure 6.6: Example USM-k clause with three variables $c_i = (\neg x_1 \cup x_2 \cup \neg x_3)$ with assignments $x_1 = 0$, $x_2 = 0$, $x_3 = 1$.

separately, including those which will be completely ignored (the segments n_{1_3} to n_{1_4} and n_{3_3} to n_{3_4}).

Then the nodes which are not connected are $c_{k_1}, c_{k_2}, c_{k_3}, n_{1_j}(n_{1_1}), n_{2_i}(n_{2_4}), n_{3_j}(n_{3_1})$. These can be connected as a single chain with every edge greater than ε by connecting the pairs $(n_{1_j}, c_{k_2}), (n_{2_i}, n_{3_j})$, and then the c_{k_1}, c_{k_3} are the end nodes. All clauses are then connected by the edge (c_{k_3}, c_{k+1_1}) for $1 \leq k \leq M-1$.

The only segments left are the switch gadgets that are not connected to a clause. These can be connected in any order provided the ends are not within ε, and we can not introduce a loop. Using this construction, the complexity of the problem can be proven.

Theorem 9. *The discrete unique set-chain matching (USM-k) problem is* ***NP****-complete.*

Proof. Given a planar 3-SAT instance φ with variables $X = \{x_1, x_2, \ldots, x_N\}$ and clauses $C = \{C_1, C_2, ..., C_M\}$, we convert it based on the method described. This

takes $O(|C|+|X|)$ and is thus polynomial. The planar 3-SAT equation φ is satisfiable if and only if there exists a polygonal line Q with nodes from the set S such that $d_F(P,Q) \leq \varepsilon$ and each point represents a unique node in Q.

Given φ is satisfiable, then for every clause, there is at least one variable which has a 'true' value. In our construction this means at least one chain does not need a point from the center of the clause, and thus we can easily find a Q such that $d_F(P,Q) \leq \varepsilon$.

If φ is unsatisfiable, then there will be at least one clause where all three variables have a false value. This means there is a clause in our construction where all three chains are in a 'false' setting, and all need a point in the clause gadget center (Figure 6.5(a)). However, since there are only two points within ε of the clause points ($c_{i_1}, c_{i_2}, c_{i_3}$ for clause C_i), at least one chain must use a point outside the clause center. This causes $d_F(P,Q) > \varepsilon$.

In the other direction, assume there exists a path Q through $S' \subset S$ such that $d_F(P,Q) \leq \varepsilon$. There must be at least one true chain at each clause, and since the three chains keep this setting back to the variable we know it had this setting at the variable. Since we also know that with the switch gadgets, the variables alternate between a true and false setting where the chain attaches. Thus, for every variable attached to a clause, it has the correct true or false setting. Therefore, if $d_F(P,Q) \leq \varepsilon$, then the current setting of each variable satisfies φ.

If no path Q does exist such that $d_F(P,Q) \leq \varepsilon$, then we know there is at least one clause where all three chains needed extra points for Q. Since the variables and switches always have a path within ε, the problem occurred in a clause. The only way this occurs is if all three chains have a 'false' setting, and similarly to the previous example, we can follow this back through the chains and see the attachment to the variables. Thus, there must also exist a clause in φ where all three variables are false.

Last, we know the problem is in **NP**. Given an instance I we can check whether $d_F(P, I) \leq \varepsilon$ in $O(mn)$ time via Theorem 1. □

Our reduction was based on the discrete Fréchet distance, but our construction also ensures that any resulting path Q is within ε of P, even along the edges. Thus, our reduction can be adapted to prove that USM-k is also **NP**-complete with the continuous Fréchet distance.

Corollary 4. The continuous Unique Set-Chain Matching (USM-k) problem based on the continuous Fréchet distance is **NP**-complete.

Proof. This proof is based on the fact that the polygonal curves P and Q are made of connected straight line segments. Given two straight lines with endpoints $a = \langle p_1, p_2 \rangle$ and $b = \langle p'_1, p'_2 \rangle$, it is straightforward to see that if $d(p_1, p'_1) \leq \varepsilon$ and $d(p_2, p'_2) \leq \varepsilon$, then under the continuous Fréchet distance $d_{\mathcal{F}}(a, b) \leq \varepsilon$.

Since both $P = \langle p_1, \ldots p_m \rangle$ and $Q = \langle q_1, \ldots q_n \rangle$ are made of straight line segments, the Fréchet distance between these segments is less than ε. There are cases where the segments are not in a one-to-one correspondence, however, since the discrete Fréchet distance is within ε in these cases, and we have straight lines, $d_{\mathcal{F}}(P, Q) \leq \varepsilon$ holds. □

CHAPTER 7

DISCRETE MAP MATCHING

We now turn our attention to the map matching problem [13] which is related to the set-chain matching problem except the goal is to find a path through a planar graph rather than a set of points. In general, this problem is more difficult than the set-chain matching problem because of the planar graph constraint, however, it also has many uses making it worthy of our attention. We will first overview some of these applications and then cover recent related work on the topic. Following, we define the discrete problem formally for the discrete Fréchet distance, and then look at the complexities of some variations of the problem similar to the set-chain matching problem.

7.1 Application

The map matching problem arose naturally out of more sophisticated GIS (Geographic Information System) software and algorithms. With the rise of excellent satellite data and the common use of GPS (Global Positioning System) in cellphones and vehicles for navigation, an important task is map routing. There are several problems dealing with map routing, but our focus will be on reconstruction tasks. Assume that we know the road network and can treat it as a planar embedded graph, and that we also have the GPS data from a car travelling. Often, this data is not accurate and may be noisy or approximate to the exact location of the vehicle. Thus, the data may not align with the roads properly.

Trying to find the most likely route of the vehicle on the road network is the standard map matching problem. We extend this analogy for modern instances where the data may be intermittent due to coverage or power constraints. A common issue with cellphones or hand-held GPS devices is limited battery life. A feasible situation is one where a user may only power on the device when near a city or when they need something immediate (such as making a call for directions), and afterwards they turn the device back off. Similarly, in remote areas, a user may only have reception near certain towns. Thus, tracking their phone is only possible when a signal is available. In these instances our data has discrete points of the lossy data. We still have a polygonal curve, but we can not depend on all edges of the line to be accurate location data. We know the node ordering, but the edges may not represent the actual path taken. So the problem is to find the most probable *simple* path of a vehicle between the recorded points on the road network.

7.2 Related Work

With respect to map matching, the problem of finding a path in a graph given a polygonal curve with respect to the Fréchet distance was first posed by Alt et. al. [13] as follows: Let $G = (V, E)$ be an undirected connected planar graph with a given straight-line embedding in $\mathbb{R}^2$ and a polygonal line P. Find a path π in G which minimizes the Fréchet distance between P and π. They give an efficient algorithm which runs in $O(pq \log q)$ time and $O(pq)$ space where p is the number of line segments of P and q is the complexity of G. Their version allows for vertices and edges to be visited multiple times, and is similar to the discrete version we cover in Section 7.4.

The recent work by Maheshwari et al. improved the running time for the map matching problem for complete graphs [12]. The original algorithm would decide it in

$O(pk^2 \log k)$ where k is the number of vertices in the graph, and their new algorithm solves it in $O(pk^2)$. This version is what we referred to as the set-chain matching problem in Chapter 6.

Map matching is an active area of research with many approaches. The two main methodologies are those based on geometric methods and ones based on Global Weight Optimization. However, the methodologies can also be classified based on the problem definition where we have local/incremental methods, global methods, and statistical methods. These can all be extended to include topological and geological conditions, current weather and traffic conditions, speed limits, and other variables that can produce more optimal routes GPS services and trip planning both online or offline [46, 47]. However, the work presented here is fully based on geometric methods. We assume that we have all of the data and want to use a geometric method (the discrete Fréchet distance) to find the best fit for the input.

There has been recent work which allowed for better performance with certain types of curves, with dual simplification for an approximate result, with bounded simplification of one of the chains, and in graphs with certain properties, [48, 49, 50]. With map matching, for the weak Fréchet distance, the bounds have been lowered further to $O(pq)$ [51], and the problems can be better defined with a smaller error bound [52].

In reality, all GPS data is discrete, and these approaches smooth the data. There are some methods optimized for low-sampling-rate data [46], but even these assume some maximum time between samples (less than five minutes). Our purpose is to analyze data where samples may be hours apart and can not be reasonably 'smoothed'.

7.3 Discrete Map Matching

The definition of discrete map matching is similar to our set-chain definition and has three variants that we will look at. We also define a similar naming scheme for the different versions using "MM" for "map matching" as opposed to "SM" for "set-chain matching" in the acronyms.

Definition 24 The Discrete Map Matching Problem.

Instance: Given a simple connected planar graph $G = (V, E)$ with a straight-line embedding in $\mathbb{R}^2$, a polygonal curve P in $\mathbb{R}^d$ ($d \geq 2$), an integer $K \in \mathbb{Z}^+$, and an $\varepsilon > 0$.

Problem: Does there exist a path Q in G with the polygonal curve formed by its edges using vertices chosen from V' where $V' \subseteq V$, such that $T \leq K$ and $d_F(P, Q) \leq \varepsilon$?

T is defined in two ways. When we are minimizing the size of the chain, $T = |Q|$. If we are minimizing the vertices in the graph used then $T = |V'|$. We look at the analogous versions of the set-chain matching problems for each of these: NMMC-k and NMMS-k. We then consider the version where the vertices in the path must be unique and only used once, which we label UMM-k. Again we note that when the vertices are unique the two minimization problems ($|Q|$,$|V'|$) are equivalent.

For reference, the naming convention is Unique/Non-unique(U/N) Map(M) Matching(M) with a k Subset/Chain(S/C). For an example explaining the difference, please refer to Figure 6.2 in Chapter 6.

7.4 NMMC-k

When focused solely on the length of Q, the problem is similar to the set-chain version (Section 6.4). The problem has an almost identical optimal substructure, so we forego the proof here. The recurrence to find the minimum size of Q (in number of vertices) is given in Equation 7.1. The actual dynamic programming algorithm is also omitted due to the similarity to Algorithm 6.9. The only difference is that each vertex v only looks at its neighbor set, $N(v)$. This is polynomial with the worst case being in a complete graph, which is equivalent to the discrete set-chain matching version. If the graph is not a complete, the complexity will be lower.

$$M[i,j] = \min \begin{cases} M[i,j\text{-}1], & \text{if } d(v_i,p_j) \leq \varepsilon, v_i \in N(v_{i\text{-}1}), M[i,j\text{-}1] \neq \emptyset \\ \min\limits_{k \in N(v_i)} (M[k,j\text{-}1]) + 1, & \text{if } d(v_i,p_j) \leq \varepsilon, v_i \in N(v_{i\text{-}1}), M[i,j\text{-}1] = \emptyset \\ \emptyset, & \text{if } d(v_i,p_j) > \varepsilon \text{ or } v_i \notin N(v_{i\text{-}1}) \end{cases} \tag{7.1}$$

Theorem 10. *The discrete Non-unique Map Matching (NMMC-k) problem where* $T = |Q|$ *is in* $\boldsymbol{P}$.

Proof. This problem is equivalent to NSMC-k with the restriction of which vertices are viable given the previous choice. Rather than looking at all previous values for the last vertex, $M[i, j-1]$, it is restricted to only looking at those vertices which have an edge between them. This is the only change to the recurrences in Equation 6.1. Thus, the problem has an identical optimal substructure and is polynomial. □

7.5 NMMS-k

This problem has proven more difficult to analyze the complexity than the other versions of the discrete map and set-chain matching problems. With the discrete Fréchet distance we show that it is **NP**-complete for general graphs (Theorem 11), and we show that the planar problem is **NP**-complete under the Hausdorff distance (Theorem 12). Even though we did show these results, we did not prove the complexity for the planar version under the discrete Fréchet distance. Due to the complexity of the other variations, we believe this problem to also be **NP**-complete, however, we leave this problem for future work.

Theorem 11. *Discrete non-unique map matching where $T = |V'|$ in general graphs is **NP**-complete.*

Proof. By a simple reduction from NSMS-k (Theorem 8) we show this is true. Given a set of points S, a polygonal curve P, $\varepsilon > 0$, and a $K \in \mathbb{Z}^+$, we build a complete graph $G = \{V, E\}$ with the vertices being the points in S, i.e. $V = S$ and $E = S \times S$.

This embedding allows all possible paths to be explored for Q, and the path it returns will be the same for both NMMS-k and NSMS-k. Thus, there exists a polygonal curve Q with nodes taken from $S' \subset S$, such that $|S'| \leq K$ and $d_F(P, Q) \leq \varepsilon$ if and only if there exists a path Q' in G such that the vertices in Q' are taken from $V' \subset V$, such that $|V'| \leq K$ and $d_F(P, Q') \leq \varepsilon$.

In the forward direction, if there is a path Q where $|S'| \leq K$ and $d_F(P, Q) \leq \varepsilon$, then there is a Q' in the graph construction where $V' \leq K$ since the same path must exist through the graph because it is complete. In the other direction, the same argument applies–if we have a path in a complete graph, then we have a polygonal curve through the points that are in the same locations as the vertices of the embedded

complete graph. We thus must only need the same points as those vertices and so $|S'| = |V'| \leq K$. □

Theorem 12. *Discrete non-unique map matching with $T = |V'|$ in planar graphs under the Hausdorff distance is **NP**-complete.*

Proof. This is a straight-forward reduction from the Hamiltonian circuit problem in grid graphs [53]. Let G_k be a grid graph with k vertices such that any vertex $v \in G_k$ is located at $v_x, v_y \in \mathbb{Z}^+$ and G_k is an induced subgraph of the infinite unit grid graph G_∞. For our construction we let $G = G_k$ and we let $P = V$ where V is the vertex set of G.

There exists a Hamiltonian Path in G_k if and only if there exists a path Q in G such that $|Q| = |V|$ and $d_H(P, Q) = 0$. If there is a Hamiltonian circuit Q' on the graph, then by definition, this path (with the start/end vertex arbitrarily chosen) has $d_H(G, Q') = 0$ since it covers every edge and vertex. Also, since $|Q'| = |V|$, $|Q| = |V|$.

Given a path Q in $G = G_k$ where $|Q| = |V|$ and $d_H(P, Q) = 0$. Note that this path must visit every edge and vertex since $d_H(P, Q) = 0$. If $|Q| = |V|$, it must only visit each vertex once, and thus the path must be a Hamiltonian circuit. □

7.6 UMM-k

Map matching with unique vertices is an interesting problem, and also one of the most relevant. In most applications related to map matching where the graph is planar, rarely would a vertex be visited multiple times. In the GPS application of a vehicle on a road network, this would be equivalent to a car visiting the same intersection multiple times. This may occur, but when trying to find the likely path

of a vehicle from the origin to the destination we can disregard self-intersecting data as unimportant overall.

Similar to the last section, we first prove the complexity of a more general version of this problem. We remove the planar constraint and prove that the problem is **NP**-complete for non-planar graphs. The complexity can be proven in a manner similar to the set-chain matching version in Section 6.6. However, a reduction from planar 3-SAT is unnecessary for this problem since we have already proven the complexity of USM-k. Simply building the complete graph on the points is sufficient. Therefore, we reduce from USM-k in this manner for a straightforward proof.

Theorem 13. *The discrete unique map matching problem is **NP**-complete for general graphs.*

Proof. We can reduce from USM-k in a straightforward fashion. Given a polygonal curve P, a set of points S, an $\varepsilon > 0$, and a $K \in \mathbb{Z}^+$. Let $G = \{V, E\}$ be the complete graph on S such that $V = S$ and $E = S \times S$.

There exists a polygonal curve Q with nodes from S such that $d_F(P, Q) \leq \varepsilon$ and $|Q| \leq K$ if and only if there exists a path Q' in G such that $d_F(P, Q') \leq \varepsilon$ and $|Q'| \leq K$.

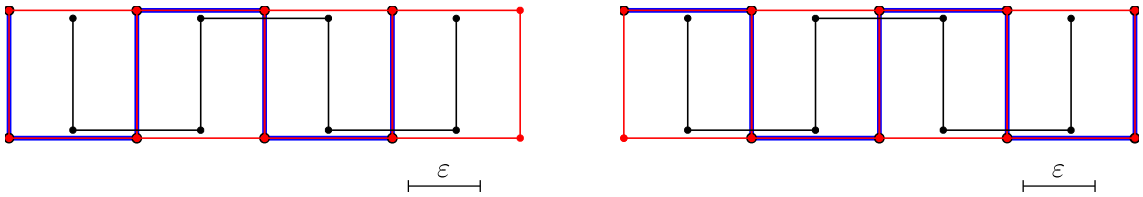
In the forward direction, if there is a path Q where $|Q| \leq K$ and $d_F(P, Q) \leq \varepsilon$, then there is a Q' in the graph version since the same path must exist through the graph because it is complete. In the other direction, the same argument applies–if we have a path in a complete graph, then we have a polygonal curve through the points that are in the same locations as the vertices of the embedded complete graph.

Finally, the problem is in **NP** since given any path Q we can verify the solution in $O(|P||Q|)$ time via Theorem 1. □

7.6.1 Planar 3-SAT Reduction

In order to show that UMM-k is **NP**-complete for planar graphs, we will reduce from planar 3-SAT, which is **NP**-complete [45], as follows.

Similar to USM-k (Section 6.6), we first show the gadgets used in the reduction and then prove the reduction from planar 3-SAT. In order to retain a true or false selection, we first show a 'chain' gadget. Figures 7.1(a) and 7.1(b) show the chain with true and false settings, respectively. Since the nodes used in G may only be used once, a path through this graph that maintains $d_F(P, Q) \leq \varepsilon$ only has two possibilities. Starting at the top left vertex a path going down is a 'true' setting and going to the right is 'false'.



(a) True Chain (b) False Chain

Figure 7.1: (a) A chain with a 'true' path. (b) A chain with a 'false' path.

Figure 7.2 shows how the chain can make a right angled turn while maintaining the true or false path. This allows the chains to be configured in a natural way with a clean layout. Note the extra edge of the graph (e_{ac}) in the corner which allows the true path to go around the vertex b. Since a false path must pass through a and b to cover p_i, it must continue up from b in order to cover p_j. The true path goes through c and has already covered p_i so it can then go through a via e_{ac} and around the outside of the corner to cover p_j and then come back down to b.

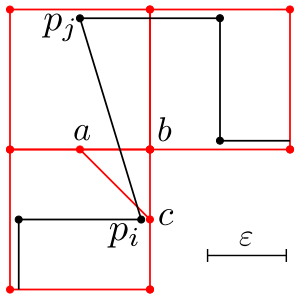
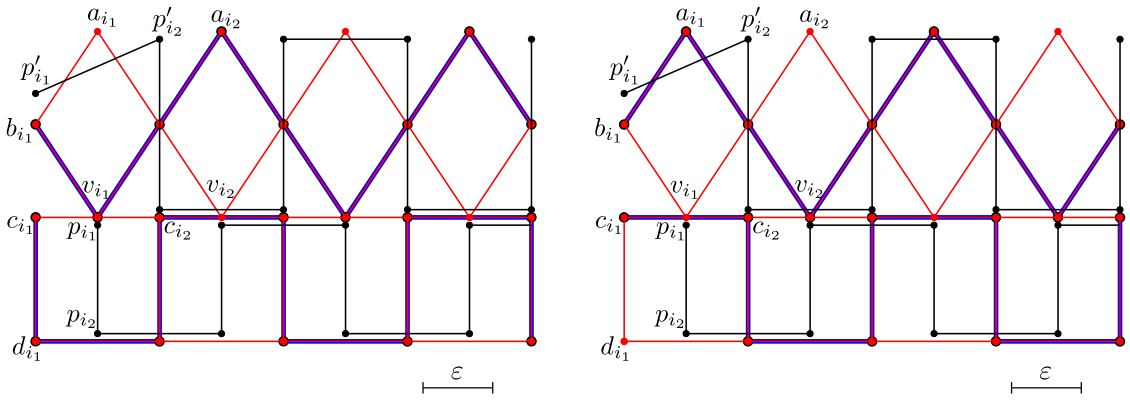


Figure 7.2: An elbow in order for the chain to make turns.

A chain is the basis of the variable gadget, but we also attach an additional diamond structure along with another polygonal curve for the diamond graph components. Figure 7.3 shows the planar graph and the two pieces of P necessary for each variable gadget. The additional diamond shapes are needed in order to force the alternation between 'true' and 'false' states. This works like the switch gadget did for USM-k. Setting a variable true or false works identical to the chain gadgets. For a variable x_i, to set the variable true, the path begins at c_{i_1} and visits vertex d_{i_1} next (Figure 7.3(a)), and to set x_i false, the path as begins at c_{i_1} and goes through vertex v_{i_1} and c_{i_2}. The chains will connect onto the a_i nodes with odd subscripts being x_i $(a_{i_1}, a_{i_3}, \dots)$, and even subscripts representing connections for $\neg x_i$ $(a_{i_2}, a_{i_4}, \dots)$. The a_i vertices connect by being shared on one edge of a chain as shown in the example of Figure 7.5. Thus, a true path in the variable does not use the a_i vertex where the chain attaches and the chain can use that edge (for a true path), but a false variable path does require the a_i which means the chain will not be able to use the edge that attaches and will also have a false path setting. We also note that all of the b_i vertices are not within ε of any of the nodes of p'_i.

For the clause gadget, we assume that three 'chains' are attached to variable gadgets and then meet at the junction shown in Figure 7.4. The clause is planar,



(a) True Variable (b) False Variable

Figure 7.3: Variable gadgets with path settings for (a) 'true' (b) 'false'. A path begins at c_{i_1} and passes through d_{i_1} for true and through v_{i_1} and c_{i_2} for false.

but our path along the edges of the graph will no longer be a polygonal curve. The discrete Fréchet distance is still valid since the distances are based on the nodes and not the edges of the curve, but this would drastically alter the continuous Fréchet distance. The vertices in the center are important to note since many of the edges curve (or turn) in the space without a vertex. This is necessary for our reduction, and thus is true for distance in the space, but does not hold for the discrete Fréchet distance based on network distance [54]. We can attach chains at one vertex to the top vertex in the variable gadgets. These chains then lead to the clause gadget. See Figure 7.5 for an example.

At a high level, we note that if a path is true in the chain it will not need to use the last edge. We will refer to the three variables in our clause as x_l, x_c, and x_r for left, center, and right, respectively. For x_l, if the path is true, then it can use the edge to s_i. If the path is false it will end at y_i and there is only one possible edge. Similarly, if x_r is true, it uses the edge to go to t_i, but a false path ends at z_i. In

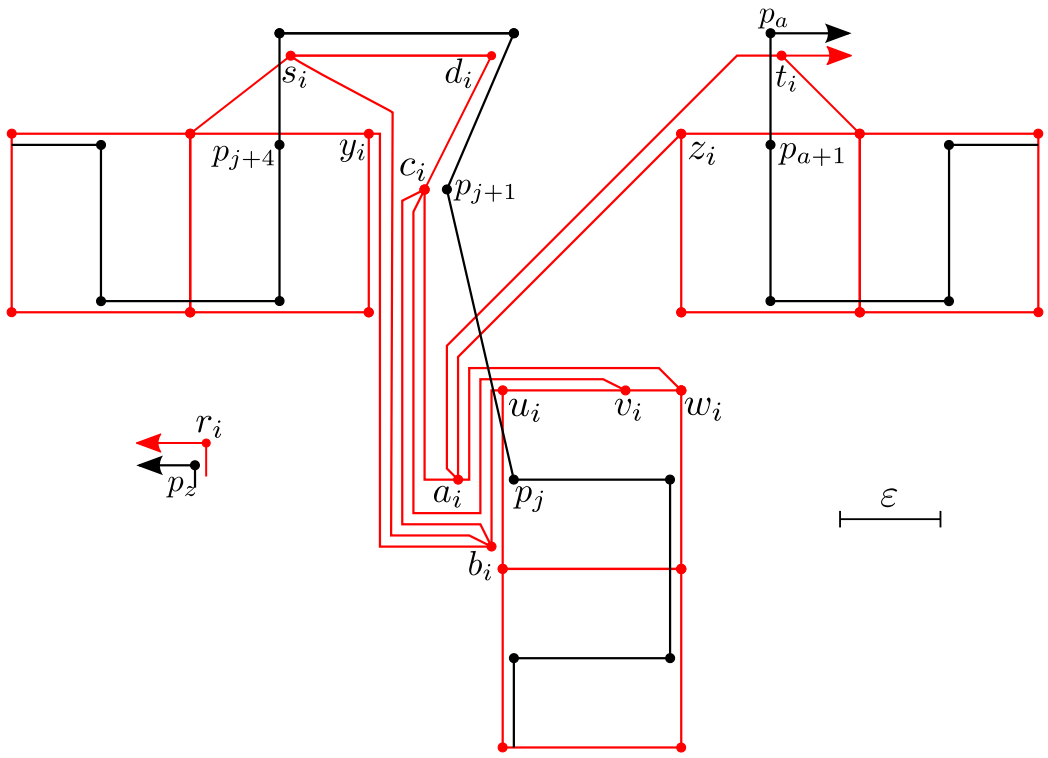


Figure 7.4: A clause with three chains meeting.

understanding the clause gadget of clause C_i, we need focus on the two vertices a_i and b_i. These two act as the Boolean 'or' operators which allow only two of the variables to be false, and thus requiring at least one to be true. Since vertices can only be used once in a path, each one (of a_i and b_i) allow only one edge coming from a variable to be used.

When x_l is false, the path ending at y_i, it must follow onto b_i and then go to s_i. Similarly, if x_r has a false path ending at z_i, it must follow onto a_i and then go to t_i. Notice that a_i and b_i are within ε of p_j. If x_c has a false path, then it can go from w_i to a_i or go to u_i and then b_i. This means that if it is false, it can go to either of the center vertices attached to x_l (b_i) or x_r (a_i). Thus, only two of them can be false. The only issue is if x_c is true because it still needs an edge to get to c_i without

crossing the edges attached to a_i and b_i. If x_c is true, then it can come from under u_i to u_i and then through v_i to c_i. Thus, our clause works as a Boolean 'or' of the three variables where at least one variable must be true.

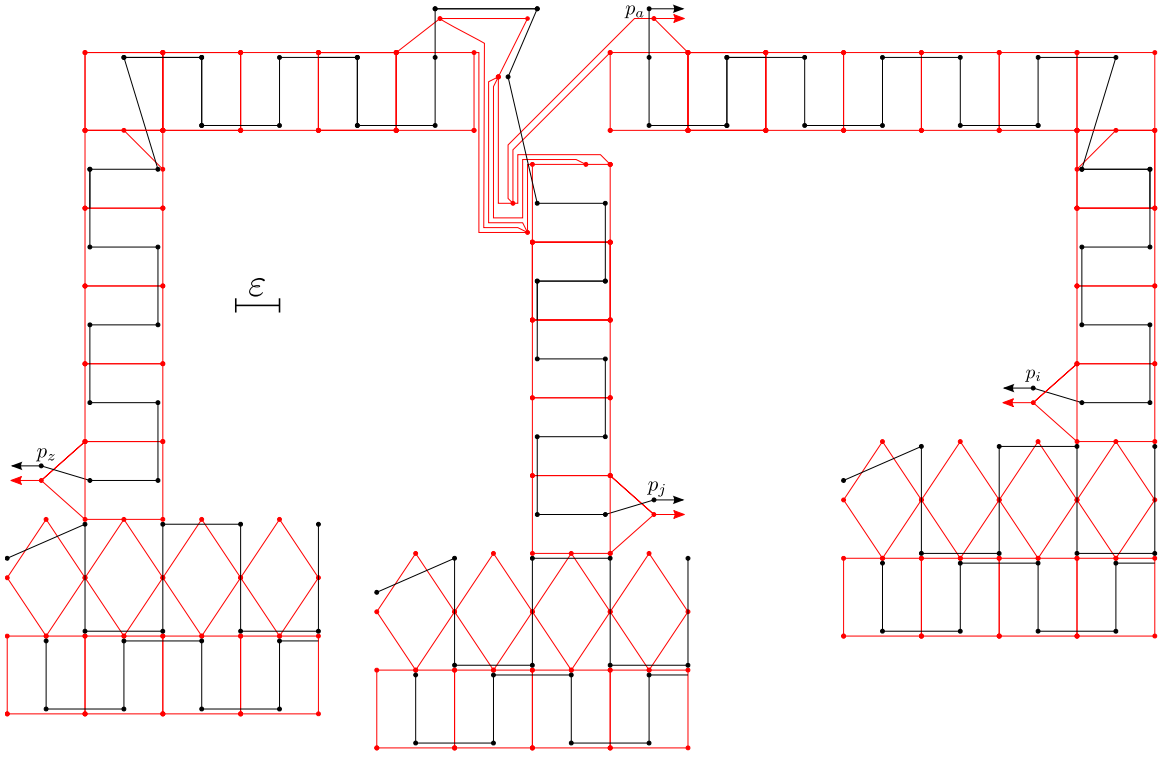


Figure 7.5: Example UMM-k clause with three variables $c_i = (\neg x_1 \cup x_2 \cup \neg x_3)$ with assignments $x_1 = 0$, $x_2 = 0$, $x_3 = 1$.

The last issue is how everything is connected so that G is planar and connected, and that P is a single line. Referring to our clause, c_i and s_i meet via d_i which allows a path to connect the two variables without changing their path settings. Looking at our example clause (Figure 7.5) we see that this leaves four polygonal curve endings (or two segments). The two exiting nodes (p_i, p_j) from x_c and x_r can simply be connected (or attached to the outside of another nested clause). This now means

each clause uses a single polygonal curve and has two nodes (p_a, p_z) to attach to other clauses or the variables along with the associated graph edges.

We can see in the example clause in Figure 7.5 that the graph and polygonal curve leave on the outside of the clause. By design, the middle and right leg connect on the interior. Thus, in a planar 3-SAT instance, if there is a nested clause, these lines (p_i, p_j) connect to the outside of the nested clause. Also note that if the nested clause is between the left leg and the center, we can simply flip the nested clause. This shows that a clause can not exist under both branches though unless that clause can be reached by way of the variables or another clause. We can also flip the right chain and it can exit the bottom. This means two graph lines have to come out of the vertex in the opposite order. Any other nesting issues can also be fixed by noting that we can attach another diamond structure on the other side of our variable gadget and have clauses on both sides of the variables.

We can then define a simple process to connect them. When saying we are connecting we mean to put an edge between the two nodes of the polygonal curves and to put an edge between the vertices of the graph. We only refer to one of the connections for simplicity. First, we begin by connecting all of the clauses. We then connect the two ends left from the clauses to the closest variable(s). Following, the two chains on each variable can be connected with the other variables consecutively. Now with this construction, the complexity of the problem can be proven.

Theorem 14. *The discrete unique map matching (UMM-k) problem is* ***NP****-complete for planar graphs.*

Proof. Given a planar 3-SAT instance φ with variables $X = \{x_1, x_2, \ldots, x_N\}$ and clauses $C = \{C_1, C_2, ..., C_M\}$, we convert it based on the method described. This takes $O(|C|+|X|)$ and is thus polynomial. The planar 3-SAT equation φ is satisfiable

if and only if there exists a polygonal line Q with nodes from the vertices in G such that $d_F(P, Q) \leq \varepsilon$ and each vertex represents a unique node in Q.

Given φ is satisfiable, then for every clause, there is at least one variable which has a 'true' value. In our construction this means at least one chain does not need a path through the two points (a_i, b_i for clause C_i), and thus we can easily find a Q such that $d_F(P, Q) \leq \varepsilon$.

If φ is unsatisfiable, then there will be at least one clause where all three variables have a false value. This means there is a clause in our construction where all three chains are in a 'false' setting, and all three variables need to pass through one of the two vertices a_i, b_i (see Figure 7.4). However, since the vertices are unique, we will not be able to find a path Q through this clause for the construction.

In the other direction, assume there exists a path Q through $V' \subset V$ such that $d_F(P, Q) \leq \varepsilon$. There must be at least one true path in a chain at each clause, and since the three chains keep this setting back to the variable we know it had this setting at the variable. Since we also know that the variables alternate between a true and false setting, the attached chain has the correct Boolean value associated with the path. Thus, for every variable attached to a clause, it has the correct true or false path setting. Therefore, if $d_F(P, Q) \leq \varepsilon$, then the current setting of each variable satisfies φ.

Suppose there is no path Q that exists such that $d_F(P, Q) \leq \varepsilon$. Since the variables and chains always have a path within ε, the problem occurred within a clause. The only way this occurs is if all three chains have a 'false' path setting and they all need a path through either a_i or b_i, but with unique vertices, at least one chain will not have a valid path. We can again follow these path settings back through the chains and see the attachment to the variables. Thus, there must also exist a clause in φ where all three variables are false.

Last, we know the problem is in **NP**. Given an instance I we can check whether $d_F(P, I) \leq \varepsilon$ in $O(mn)$ time via Theorem 1. □

Unlike USM-k, the clause gadget for this reduction (Figure 7.4) requires the path to follow the graph through edges that are farther than ε from P, and therefore we can not state anything about the complexity of UMM-k based on the standard continuous Fréchet distance. There may be modifications to our reduction that allow it to work, but we have not pursued any since this is not the focus of our investigation, and so we leave this as another open problem for future research.

CHAPTER 8

SOFTWARE

As previously mentioned, the majority of software systems for aligning and comparing protein backbones use the RMSD measure. Thus, we have created a software library called The Fréchet-based Protein Alignment & Comparison Toolkit (FPACT), which uses the alignment, comparison, and simplification algorithms (including CPS-3F$^+$) based on the discrete Fréchet distance [11].

8.1 FPACT

To facilitate research using the discrete Fréchet distance we have also created a set of libraries to run any of our algorithms based on the discrete Fréchet distance. The FPACT (The Fréchet-based Protein Alignment & Comparison Toolkit) libraries were designed for easy access to the algorithms by being modular and protein file format independent. The toolkit includes methods and classes such as the discrete Fréchet distance, ALIGN (Algorithm 3.1), SIMPLIFY (Algorithm 4.2), versions of CPS-3F$^+$ optimized for space or time efficiency (Algorithm 4.3), the CPS-3F$^+$ backtracking algorithm (Algorithm 4.4), and some other utility functions. The libraries will be updated with any future algorithms or results as well. All libraries are written and available in both C# and Python with Numpy.

We have also implemented a simple web-based application which uses these libraries. The web-based application runs within the Silverlight framework, and can be used in any browser supporting the Silverlight or Moonlight runtime. The software is available to the public for general use, thus providing the ability to align, compare,

and simplify protein backbones with the discrete Fréchet distance without directly using the libraries [11].

FPACT, the web application, and relevant documentation about the research, can be found at the website http://www.cs.montana.edu/∼timothy.wylie/frechet/.

8.2 Biological Libraries

Our libraries use standard numerical arrays, and are independent of any biological data structures or file formats. In order to use FPACT with standard bioinformatics based files, such as PDB, other software libraries must also be used. Most of our work was done using SPADE (the Structural Proteomics Application Development Environment) [55], although the software has since been abandoned in favor of more modern libraries. One of the most complete and straightforward toolkits for biological applications written for Python is BioPython [56]. Other software libraries in the same vein are also available for most of the popular programming languages.

CHAPTER 9

CONCLUSION

In this dissertation we have covered some applications and many problems related to them which are based on the discrete Fréchet distance. Through this investigation we have extended what is known of the discrete Fréchet distance and found solutions to many of the problems.

In the area of protein backbone structure alignment, we created the first alignment algorithm based on the discrete Fréchet distance and proved that it is a factor-2 approximation to the optimal alignment problem. In doing so, we also empirically reinforced the evidence that the discrete Fréchet distance has many benefits for protein backbone comparisons over the standard root mean square deviation.

In the area of visualization between two large polygonal curves, we relied heavily on the chain pair simplification problem under the discrete Fréchet distance (CPS-3F). Although the complexity of CPS-3F is still unknown, we made great strides in understanding many properties of the problem. Using these, we devised a heuristic algorithm and two factor-2 approximation algorithms. The approximation algorithms defined the moving cost, and also many other properties, and were obtained with efficient dynamic proprramming solutions. Further, we looked at chain pair simplification problem under the Hausdorff distance (CPS-2H), which is known to be **NP**-complete. We proved that even based on the moving cost, the problem is still **NP**-complete.

A couple of other applications covered include the discrete map and set-chain matching problems which have broad applicability in GIS systems and wireless networking. Not only did we address the standard problems, but proposed useful variations of both. Two of these optimization problems were shown to be polynomial via

dynamic programming. We proved that three versions were **NP**-complete. There is one problem that remains open, but while working on it, we showed that two related problems are **NP**-complete.

This dissertation covers work that has already been published as well as research that is awaiting publication or has not been sent to publication yet. Many of these applications are extremely important and our research has only highlighted many of the uses for, and benefits of, the discrete Fréchet distance.

REFERENCES CITED

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

[2] Tim Wylie, Jun Luo, Binhai Zhu. A practical solution for aligning and simplifying pairs of protein backbones under the discrete Fréchet distance. *Proceedings of the 2011 international conference on Computational science and its applications - Volume Part III*, ICCSA'11, pages 74–83, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Tim Wylie, Binhai Zhu. A polynomial time solution for protein chain pair simplification under the discrete Fréchet distance. *Proceedings of the 2012 International Symposium on Bioinformatics Research and Applications*, ISBRA'12, pages 287–298, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884 - 1940)*, 22(1):1–72, December 1906. doi:10.1007/BF03018603.

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

[6] Thomas Eiter, Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Information Systems Department, Technical University of Vienna, 1994.

[7] Sergey Bereg, Minghui Jiang, Wencheng Wang, Boting Yang, Binhai Zhu. Simplifying 3d polygonal chains under the discrete Fréchet distance. *Proceedings of the 8th Latin American conference on Theoretical informatics*, LATIN'08, pages 630–641, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] Tim Wylie, Binhai Zhu. Protein chain pair simplification under the discrete Fréchet distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2013.

[9] Axel Mosig, Michael Clausen. Approximately matching polygonal curves with respect to the Fréchet distance. *Computational Geometry: Theory and Applications*, 30(2):113–127, Feb 2005.

[10] R. Sriraghavendra, Karthik K., C. Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. *Proceedings of the Ninth*

International Conference on Document Analysis and Recognition - Volume 01, ICDAR '07, pages 461–465, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Tim Wylie. FPACT: The Fréchet-based protein alignment & comparison toolkit, 2012. http://www.cs.montana.edu/∼timothy.wylie/frechet.

[12] Anil Maheshwari, Jrg-Rdiger Sack, Kaveh Shahbaz, Hamid Zarrabi-Zadeh. Staying close to a curve. *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry*, CCCG'11, 2011. August 10-12, 2011.

[13] Helmut Alt, Alon Efrat, Günter Rote, Carola Wenk. Matching planar maps. *J. Algorithms*, 49(2):262–283, November 2003.

[14] Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, Carola Wenk. Fréchet distance for curves, revisited. *Proceedings of the 14th conference on Annual European Symposium - Volume 14*, ESA'06, pages 52–63, London, UK, 2006. Springer-Verlag.

[15] Helmut Alt, Michael Godau. Measuring the resemblance of polygonal curves. *Proceedings of the eighth annual symposium on Computational geometry*, SoCG'92, pages 102–109, New York, NY, USA, 1992. ACM.

[16] Michael Godau. A natural metric for curvescomputing the distance for polygonal chains and approximation algorithms. *Proceedings of the 8th annual symposium on Theoretical aspects of computer science*, STACS'91, pages 127–136, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[17] Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *CoRR*, abs/1204.5333, 2012.

[18] Felix Hausdorff. *Grundzge der mengenlehre*. Von Veit, Leipzig, 1914.

[19] J.R. Munkres. *Topology*. Prentice Hall, Incorporated, 2000.

[20] T. K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968. Russian Kibernetika 4(1):81-88 (1968).

[21] Meinard Müller. *Information Retrieval for Music and Motion*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[22] R. Sriraghavendra. *Language-neutral Algorithms for Partial Search on Online Handwritten Text*. Doctoral Dissertation, Indian Institute of Science, Jul 2007.

[23] Helmut Alt, Bernd Behrends, Johannes Blömer. Approximate matching of polygonal shapes (extended abstract). *Proceedings of the 7th Annual Symposium on Computational Geometry*, SoCG '91, pages 186–193, New York, NY, USA, 1991. ACM.

[24] Helmut Alt, Christian Knauer, Carola Wenk. Matching polygonal curves with respect to the Fréchet distance. *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '01, pages 63–74, London, UK, UK, 2001. Springer-Verlag.

[25] Carola Wenk. *Shape Matching in Higher Dimensions*. Doctoral Dissertation, Freie Universitaet Berlin, 2002.

[26] S. B. Needleman, C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[27] C A Mauzy, M A Hermodson. Structural homology between rbs repressor and ribose binding protein implies functional similarity. *Protein Science*, 1(7):843–849, 1992.

[28] C. Strauss A. Oritz, O. Olmea. Mammoth (matching molecular models obtained from theory): an automated method for model comparison. *Protein Science*, 11:2606–2621, 2002.

[29] Loredana Lo Conte, Bart Ailey, Tim J. P. Hubbard, Alexey G. Murzin, Cyrus Chothia. Scop: a structural classification of proteins database. *Nucleic Acids Research*, 28:257–259, 2000.

[30] L. Holm, C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233:123–138, 1993.

[31] Liisa Holm, Jong Park. Dalilite workbench for protein structure comparison. *Bioinformatics*, 16(6):566–567, 2000.

[32] C.A. Orengo, A.D. Michie, D.T. Jones, M.B. Swindells, J.M. Thornton. Cath: A hierarchic classification of protein domain structures. *Structure*, 1(5):1093–1108, 1997. Using secondary structures to measure the geometry of a protein.

[33] I. Shindyalov, P. Bourne. Protein structure alignment by incremental combinatorial extension (ce) of the optimal path. *Protein Engineering*, 11:739–747, 1998.

[34] Chi-Ren Shyu, Pin-Hao Chi, Grant Scott, Dong Xu. Proteindbs: a real-time retrieval system for protein structure comparison. *Nucleic Acids Research*, 32(Web Server issue):W572–575, 2004.

[35] W. Taylor, C. Orengo. Protein structure alignment. *Journal of Molecular Biology*, 208:1–22, 1989.

[36] Jinn-Moon M. Yang, Chi-Hua H. Tung. Protein structure database search and evolutionary classification. *Nucleic Acids Research*, 34(13):3646–3659, 2006.

[37] Zhixiang Chen, Bin Fu, Binhai Zhu. The approximability of the exemplar breakpoint distance problem. *Proceedings of the 2nd International Conference on Algorithmic Aspects in Information and Management (AAIM'06)*, Volume 4041 series *Lecture Notes in Computer Science*, pages 291–302. Springer, 2006.

[38] Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, 1975.

[39] Hiroaki Sakoe, Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

[40] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, Carola Wenk. On map-matching vehicle tracking data. *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 853–864. VLDB Endowment, 2005.

[41] Gautam K. Das, Robert Fraser, Alejandro Lòpez-Ortiz, Bradford G. Nickerson. On the discrete unit disk cover problem. *Proceedings of the 5th international conference on WALCOM: algorithms and computation*, WALCOM'11, pages 146–157, Berlin, Heidelberg, 2011. Springer-Verlag.

[42] Robert Fraser, Alejandro Lòpez-Ortiz. The within-strip discrete unit disk cover problem. *Proceedings of the 24th Canadian Conference on Computational Geometry, CCCG 2012, Charlottetown, Prince Edward Island, Canada, August 8-10, 2012*, CCCG'12, pages 53–58, 2012.

[43] Rashmisnata Acharyya, Manjanna B., Gautam K. Das. Unit disk cover problem. *CoRR*, abs/1209.2951, 2012.

[44] Nabil H. Mustafa, Saurabh Ray. Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44(4):883–895, December 2010. doi:10.1007/s00454-010-9285-9.

[45] David Lichtenstein. Planar Formulae and Their Uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[46] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, Yan Huang. Map-matching for low-sampling-rate gps trajectories. *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information*

Systems, GIS '09, pages 352–361, New York, NY, USA, 2009. ACM. doi:10.1145/1653771.1653820.

[47] Hong Wei, Yin Wang, George Forman, Yanmin Zhu. Map matching by Fréchet distance and global weight optimization. Technical Report SJTU_CS_TR_201302001, Department of Computer Science and Engineering, Shanghai Jiao Tong University, 2013.

[48] Kevin Buchin, Maike Buchin, Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 645–654, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[49] Daniel Chen, Anne Driemel, Leonidas J. Guibas, Andy Nguyen, Carola Wenk. Approximate map matching with respect to the Fréchet distance. Matthias Müller-Hannemann, Renato Fonseca F. Werneck, editors, *Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments*, ALENEX'11, pages 75–83. SIAM, 2011.

[50] Anne Driemel, Sariel Har-Peled, Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012.

[51] Daniel Chen, Leonidas J. Guibas, Qixing Huang, Jian Sun. A faster algorithm for matching planar maps under the weak Fréchet distance. Unpublished, December 2008.

[52] Carola Wenk, Randall Salas, Dieter Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. *18th International Conference on Scientific and Statistical Database Management*, SSDBM'06, pages 379–388. IEEE Computer Society, 2006.

[53] Alon Itai, Christos H. Papadimitriou, Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982. doi:10.1137/0211056.

[54] Chenglin Fan, Jun Luo, Binhai Zhu. Fréchet-distance on road networks. *Proceedings of the 9th international conference on Computational Geometry, Graphs and Applications*, CGGA'10, pages 61–72, Berlin, Heidelberg, 2011. Springer-Verlag. doi:10.1007/978-3-642-24983-9_7.

[55] Deacon J. Sweeney, Gerald M. Alter, Michael L. Raymer. Introducing SPADE, the Structural Proteomics Application Development Environment. Ohio Collaborative Conference on Bioinformatics (OCCBIO'08), June 2008.

[56] Peter J A Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, et al. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009.

[57] Alexander Wolff. A simple proof for the NP-hardness of edge labeling. Technical Report W-SPNPH-00, Institut für Mathematik und Informatik, Universität Greifswald, 2000.

[58] Minghui Jiang, Sergey Bereg, Zhongping Qin, Binhai Zhu. New bounds on map labeling with circular labels. *Proceedings of the 15th international conference on Algorithms and Computation*, ISAAC'04, pages 606–617, Berlin, Heidelberg, 2004. Springer-Verlag. LNCS 3341. doi:http://dx.doi.org/10.1007/978-3-540-30551-4_53.

[59] Minghui Jiang. *Map Labeling with Circles*. Doctoral Dissertation, Montana State University, 2005.

[60] Tim Wylie, Binhai Zhu. Discretely following a curve. *Short Abstract for Computational Geometry: Young Researchers Forum*, CG:YRF'12, pages 33–34, 2012.

[61] Hee-Kap Ahn, Christian Knauer, Marc Scherfenberg, Lena Schlipf, Antoine Vigneron. Computing the discrete Fréchet distance with imprecise input. Otfried Cheong, Kyung-Yong Chwa, Kunsoo Park, editors, *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC 2010)*, Volume 6507 series *Lecture Notes in Computer Science*, pages 422–433. Springer-Verlag, Berlin/Heidelberg, Germany, 2010.

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

[63] M. R. Garey, D. S. Johnson, L. Stockmeyer. Some simplified np-complete problems. *Proceedings of the sixth annual ACM symposium on Theory of computing*, STOC '74, pages 47–63, New York, NY, USA, 1974. ACM. doi:10.1145/800119.803884.

[64] R. J. Fowler, M. S. Paterson, S. L Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133–137, jun 1981.

[65] N. Megiddo, K. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984.

[66] Franz Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, September 1991. doi:10.1145/116873.116880.

[67] Axel Mosig. *Efficient Algorithms for Shape and Pattern Matching*. Doctoral Dissertation, University of Bonn, Germany, 2004.

[68] Xiaohua Xu, Zhu Wang. Wireless coverage via dynamic programming. Yu Cheng, DoYoung Eun, Zhiguang Qin, Min Song, Kai Xing, editors, *Wireless Algorithms, Systems, and Applications*, Volume 6843 series *Lecture Notes in Computer Science*, pages 108–118. Springer Berlin Heidelberg, 2011.

[69] Minghui Jiang. Inapproximability of maximal strip recovery. *Theoretical Computer Science*, 412(29):3759 – 3774, 2011. doi:10.1016/j.tcs.2011.04.021.

[70] Lawrence Rabiner, Biing-Hwang Juang. *Fundamentals of speech recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[71] Simina Vasilache, Nazanin Mirshahi, Soo-Yeon Ji, James Mottonen, Donald J. Jacobs, Kayvan Najarian. A signal processing method to explore similarity in protein flexibility. *Advances in Bioinformatics*, 2010, 2010.

[72] Piotr Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. *Proceedings of the eighteenth annual symposium on Computational geometry*, SCG '02, pages 102–106, New York, NY, USA, 2002. ACM. doi:10.1145/513400.513414.

[73] Binhai Zhu. Protein local structure alignment under the discrete Fréchet distance. *Journal of Computational Biology*, 14(10):1343–1351, 2007.

[74] Maike Buchin. *On the Computability of the Fréchet Distance Between Triangulated Surfaces*. Doctoral Dissertation, Free University Berlin, Institute of Computer Science, 2007.

[75] Anne Driemel, Sariel Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 318–337. SIAM, 2012.

[76] Tim Wylie, Binhai Zhu. Discretely following a curve. *Preparation*, 2013.