APPLICATION OF GEOMETRIC GRAPH DISTANCES IN MACHINE LEARNING CLASSIFICATION

AN HONORS THESIS

SUBMITTED ON THE 29TH DAY OF APRIL, 2024

TO THE DEPARTMENT OF COMPUTER SCIENCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

OF THE HONORS PROGRAM

OF NEWCOMB-TULANE COLLEGE

TULANE UNIVERSITY

FOR THE DEGREE OF

BACHELOR OF SCIENCE

WITH HONORS IN THE DEPARTMENT OF COMPUTER SCIENCE

BY

[illegible]

WILLIAM RODMAN

APPROVED:

[illegible]

DR. CAROLA WENK
Director of Thesis

[illegible]

DR. RAMGOPAL METTU
Second Reader

Abstract

Geometric graph theory, prevalent in computer science for structuring networks like the World Wide Web and digital road maps, lacks computationally efficient methods for measuring distances between graphs. This thesis investigates the traversal distance algorithm's potential in machine learning classification, specifically in enhancing the precision of the k-nearest neighbors (k-NN) model for classifying geometric graphs represented as English letters. Prior research has explored various algorithms for this purpose, the hypothesis being, the traversal distance offers computational efficiency despite its asymmetry. Employing a dataset of English letters, the research tests the hypothesis that a k-NN model, integrated with traversal distance, could achieve high classification precision. Through comparative analysis with the graph edit distance (GED), the traversal distance's effectiveness in classification tasks is established. Demonstrating the traversal distance's potential application in supervised machine learning.

Contents

Acknowledgments

I want to extend my gratitude to the individuals who have guided and supported me throughout my research. Their expertise, encouragement, and mentorship played a important role in shaping this thesis.

First and foremost, thank you to Dr. Carola Wenk, Professor at Tulane University's Department of Computer Science. Dr. Wenk introduced me to research in my freshman year of college as a research assistant. I have gained significant research experience over the last three years through my involvement in the project 'A Unified Framework for Geometric and Topological Signature-Based Shape Comparison,' supported by the National Science Foundation under grant number NSF-CCF 2107434. This research laid the foundation for my honors thesis, where I have continued to work with Dr. Wenk as my primary thesis advisor. Next, I would like to thank Erfan Hosseini, a Computer Science PhD candidate at Tulane University, advised by Dr. Wenk. Erfan has helped me countless times, both in my time as a research assistant and throughout the research for my honors thesis. Thank you to Dr. Ramgopal Mettu, Professor in the Department of Computer Science at Tulane University, for his role as my second thesis reader. I also extend my thanks to Dr. Sushovan Majhi, a Post-doc researcher at the University of California, Berkeley. Working with Dr. Majhi has deepened my understanding of the Graph Edit Distance. His contribution of the English letter dataset was an important component of this thesis. My gratitude also goes to Dr. Liz Munch and Dr. Sarah Percival, faculty at Michigan State University, for their advice and provision of the plant leaf dataset to the research group at Tulane University.

In addition to my academic mentors, the support of my family has also been a source of motivation. A special thank you to my father, Rick Rodman, who supported me throughout my bachelor's degree. His career at NASA Goddard Space Flight Center sparked my interest in mathematics and computer science. To my brother, Sander Rodman, an undergraduate at Cornell University studying information science and statistics. And finally, to my mother and stepfather, Laura and Todd Ecker.

1 Introduction

Geometric graph theory has become a significant concept in modern mathematics. Its relevance extends into the field of computer science, where it plays a crucial role in defining and structuring complex networks, including the World Wide Web and digital road mapping systems like Apple Maps [2]. The widespread application of geometric graphs in computational geometry has garnered interest from research organizations, including the National Science Foundation. This interest has led to funding for academic research focused on advancing methods for comparing, measuring, and efficiently storing geometric graphs.

The measurement of distance between two geometric graphs is a practical comparison in geometric graph theory. However, the comparison faces challenges since there exists no closed-form solutions for computing the spatial distance between graphs. This challenge has prompted the development of a variety of algorithms designed to approximate this distance. The focus of this thesis is to investigate the traversal distance algorithm, examining its advantages, disadvantages, and applications [1]. One advantage of this algorithm is its computational efficiency; it has a polynomial time complexity, contrasting the NP-Hard complexity of the graph edit distance (GED). However, a notable limitation is its asymmetry, an issue that GED avoids [7].

This thesis will test the application of the traversal distance in the classification of geometric graphs, a challenge within supervised machine learning. Supervised learning, a subfield of computer science and artificial intelligence, has gained significant traction, particularly following OpenAI's release of ChatGPT [5]. The central hypothesis of this research is that a k-nearest neighbors (k-NN) machine learning model, when integrated with the traversal distance as its distance metric, will prove precise in classifying geometric graphs. This hypothesis will be empirically tested using a dataset of English letters represented as geometric graphs.

2 Introducing the Traversal Distance

In this chapter, the discussion begins by explaining what geometric graphs are; the inputs needed to compute the traversal distance. Next, the chapter introduces a simpler way to measure the distance between two curves, known as the weak Fréchet distance. This topic acts as a stepping stone, helping to better understand the more complex traversal distance. Finally, the advantages and disadvantages of using the traversal distance are discussed. It is compared with other methods of measuring distances in geometric graphs, like the graph edit distance (GED), to gauge its usefulness.

2.1 Properties of Geometric Graph Theory

Geometric graph theory exists within the broader field of graph theory, focused on the study of graphs embedded in a geometric space. In the case of this thesis, graphs are embedded in Euclidean planes, where vertices represent coordinate points and edges signify line segments between these points.

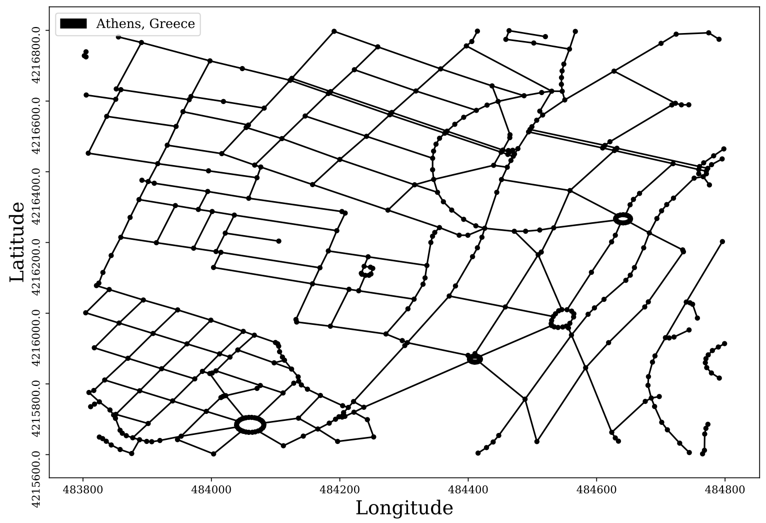


Figure 1: Geometric graph capturing network of roads in Athens, Greece [6].

This discipline extends into numerous applications in computer science. Consider a digital road map, a familiar tool used while driving a car. In this context, the roads can be thought of as the edges of a geometric graph, while the intersections where these roads meet are the vertices. While traditional road maps reference geographic coordinates to pinpoint locations, digital road maps, when stored as geometric graphs, contain embedded coordinates that correspond to locations [9].

Definition 1. *Let a geometric graph G be defined as a pair $G = (V, E)$, where V is the set of vertices, with each vertex $v \in V$ corresponding to a pair of two-dimensional coordinates (x, y) in the Euclidean plane. Then the set V is defined as:*

$$V = \{1, 2, \ldots, n\} \quad \textit{where} \quad n \in \mathbb{N} \quad \textit{such that} \quad v_i \mapsto (x_i, y_i)$$

E is the set of edges, ensuring G is a undirected geometric graph. Each edge $e \in E$ defines a line segment bounded by two vertices. An edge e comprised of vertices v_i and v_j is denoted as e_k. Then the set E is defined as:

$$E = \{e_k \mid (v_i, v_j) \in V \times V \textit{ and } i < j\} \quad \textit{such that} \quad e_k = (v_i, v_j)$$

The implementation of this geometric graph within a Python program takes on a slightly different form. The Python program in this thesis stores geometric graphs using two dictionaries. The first dictionary, which will be referred to as `nodes`, contains all the vertices of the graph. In this `nodes` dictionary, the keys are used for vertex identification, and the values are tuples containing the coordinates of each vertex. The `nodes` dictionary can be written as:

$$nodes = \{1 : (x_1, y_1), 2 : (x_2, y_2), \ldots n : (x_n, y_n)\}$$

The second dictionary, `nodeLinks`, stores all information pertaining to the edges of the geometric graph. In this dictionary, the keys contain vertices that form part of an edge's line segment. The values associated with these keys are lists, each containing the neighboring vertices that each completes an edge's line segment. The `nodeLinks` dictionary can be written as [13]:

$$nodeLinks = \{1 : Adj(1), 2 : Adj(2), \ldots n : Adj(n)\}$$
$$Adj(i) = \{j \mid (v_i, v_j) \in E\}$$

The configuration of geometric graphs in the Python program, as detailed in a later section of this chapter, reduces the time for searching for all nodes adjacent to a given node [9].

2.2 Weak Fréchet Distance

Before mentioning the traversal distance, it is instructive to first understand the weak Fréchet distance, a metric for measuring the distance between two curves. The rationale for starting with the weak Fréchet distance is that it presents a simpler problem, which provides a foundation for grasping the more generalized traversal distance problem.

Letting C_1 and C_2 be defined as two polygonal curves, where each curve has a first and last point. Additionally, let free-space be a subspace of the parameter spaces for C_1 and C_2, defined by the value ϵ. Essentially, the weak Fréchet distance is the value ϵ, the parameter of free-space spanning entirely between the first and last points of C_1 and C_2. More strictly, the value ϵ is a weak Fréchet distance if and only if there exists a continues path across the free-space, such that the path starts at the free-space for the first points of C_1 and C_2 and ends at the free-space for the last points of C_1 and C_2.

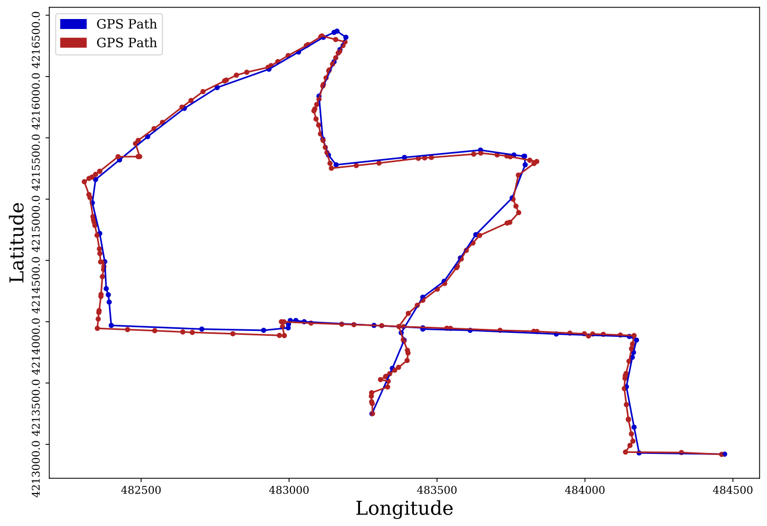


Figure 2: GPS coordinate paths as polygonal curves [14].

To illustrate, consider the scenario in Figure 2 comparing a GPS-tracked route to a hiking trail stored digitally. Here, the weak Fréchet distance would be used to determine the greatest deviation of the GPS route from the hiking trail, with the GPS trajectory and the hiking trail representing the two curves in question.

Definition 2. *Given two curves C_1 and C_2 that exist in the Euclidean plane. Let $C_1 : [0,1] \to \mathbb{R}^2$, $C_2 : [0,1] \to \mathbb{R}^2$, and $\|\cdot\|$ denote the Euclidean norm. Then the weak Fréchet distance $\delta_F(C_1, C_2)$ is defined as:*

$$\delta_F(C_1, C_2) := \inf_{\alpha,\beta} \max_{t\in[0,1]} \|C_1(\alpha(t)) - C_2(\beta(t))\|,$$

where $\alpha : [0,1] \to [0,1]$ and $\beta : [0,1] \to [0,1]$ range over continuous parameterizations with $\alpha(0) = 0$, $\alpha(1) = 1$, $\beta(0) = 0$, and $\beta(1) = 1$.

This introduction of the weak Frechet distance equation sets the stage for the definition of the traversal distance in section 2.4 [1].

2.3 Free-Space

Before defining the traversal distance and algorithm, it is necessary to introduce the formal definition of free-space and the data structure used to store free-space. In this thesis, free-space will be referred to as FS_ϵ.

Definition 3. *Given the distance threshold $\epsilon \geq 0$. Let FS_ϵ be the region in the parameter space that consists of all edge pairs for C_1 and C_2 for distance at most ϵ. Then FS_ϵ is defined as:*

$$FS_\epsilon = \{(\theta, \gamma) \in [0,1]^2 \mid \|C_1(\theta) - C_2(\gamma)\| \leq \epsilon\}$$

Each combination of edges within C_1 and C_2 is designated a square free-space cell [1]. Figure 3 shows an example of a pair of edges, e_i and e_j, taken from the polygonal curves C_1 and C_2, respectively. Figure 4 illustrates the parameter space of e_i and e_j, highlighting the free-space in white when $\epsilon = 6$ [13].

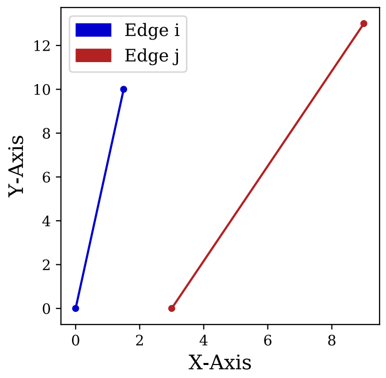


Figure 3: Pair of distinct edges [13].

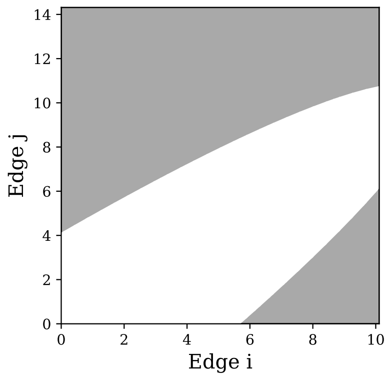


Figure 4: Corresponding free-space cell when $\epsilon = 6$ [13].

While Definition 2 describes the parametrization of space with curves, the weak Fréchet distance algorithm actually represents FS_ϵ using discrete space [1]. In this case, a free-space cell consists of four cell boundaries, each representing a wall of the square free-space cell. Each cell boundary within a free-space cell stores the starting and ending points, denoted $(start, end)$, of free-space along a cell wall. These eight points collectively form a polygon that contains the cell's free-space [13]. Figure 5 illustrates this, using the same pair of edges from Figure 3. Figure 6 shows how the corresponding free-space cell from Figure 5 is stored in the cell boundary data structure.

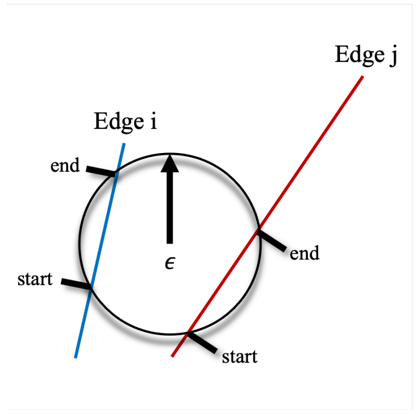


Figure 5: Cell boundary starting and ending points calculated based on value ϵ.

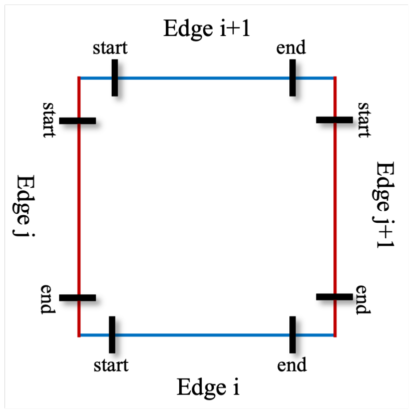


Figure 6: Cell boundary starting and ending points inside a free-space cell.

As will be shown in the next section, the use of cell boundaries can be applied more generally to two geometric graphs, G_1 and G_2 [1]. The Python program stores `cell_boundaries` in the form of a dictionary [13]:

$$cell\_boundaries = \{(e, v) : cellBoundary(e, v)) \mid P\}$$

$$cellBoundary(e_i, v_j) = (start, end) \quad \text{where} \quad 0 \leq start < end \leq 1$$

$$P = (v \in V_1 \text{ and } \; e \in E_2) \text{ or } (v \in V_2 \text{ and } \; e \in E_1)$$

2.4 Traversal Distance and Algorithm

The traversal distance, defined in chapter 1, should be thought of as a more general form of the weak Fréchet distance. While the weak Fréchet distance is concerned with measuring the distance between two curves C_1 and C_2, the traversal distance extends this concept to measure the distance between two geometric graphs G_1 and G_2.

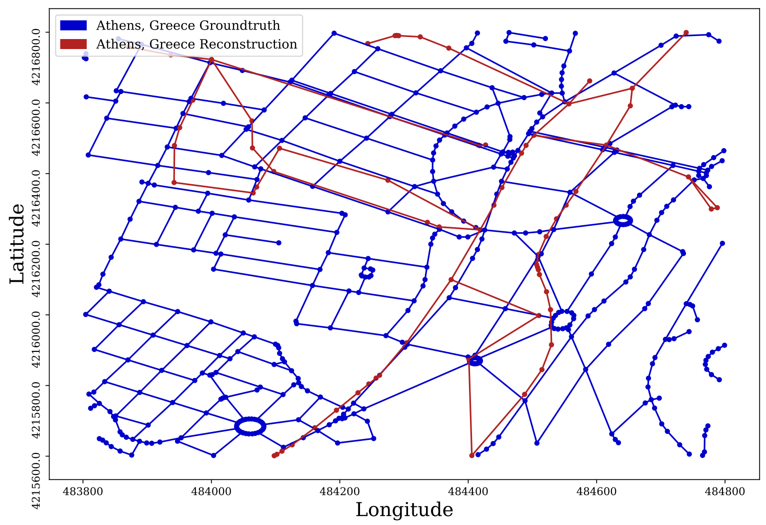


Figure 7: Two networks of roads in Athens, Greece as geometric graphs [6].

To understand this, consider the traversal distance from G_1 to G_2 as the maximum distance required to cover any point on the edges of G_1, while simultaneously traversing the entirety of G_2 in a continuous fashion. A key distinction between the traversal distance and the weak Fréchet distance is their symmetry properties. The weak Fréchet distance is symmetric, meaning the distance from C_1 to C_2 is identical to that from C_2 to C_1. In contrast, the traversal distance is asymmetric, such that the distance from G_1 to G_2 may differ from the distance from G_2 to G_1, since the entire traversal of G_2 in a continuous fashion is not required.

Definition 4. *Given two geometric graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Define the traversal distance δ_T from G_2 to G_1 as:*

$$\delta_T(G_1, G_2) = \inf_{f,g} \max_{t\in[0,1]} \|f(t) - g(t)\|$$

where f traverses entirely over G_2 and g traverses partially over G_1.

After defining the traversal distance equation, it is important to note that due to its nature as an infimum, the traversal distance algorithm is computationally expensive. This characteristic creates an opportunity for a less expensive algorithm for approximating the traversal distance, offering a practical solution for evaluating this metric in real-world applications [1].

The algorithm designed to decide traversal distance is comprised of three components. The first component, a dynamic program, populates `cell_boundaries` for a given ϵ. The second component verifies that G_2 has been completely traversed by the first algorithm. This is done by projecting `cell_boundaries` onto G_2, if the projection covers the entirety of G_2, then the verification is true. When verification is true, these two algorithms are responsible for deciding whether $\delta_T(G_1, G_2) \leq \epsilon$. To approximate the infimum of the traversal distance, a binary search algorithm is implemented. This algorithm minimizes the distance to meet specific precision criteria, ultimately yielding an approximation ϵ where $\delta_T(G_1, G_2) \leq \epsilon \ \wedge \delta_T(G_1, G_2) \approx \epsilon$ [13].

Step 1: Compute the Cell Boundaries. Defined in Algorithm 1, the dynamic program used to compute `cell_boundaries` is a depth-first search (DFS) algorithm `DFSTraversalDist`. This algorithm writes to `cell_boundaries` while traversing all cell boundaries in FS_ϵ.

Algorithm 1 DFSTraversalDist

```
Initialize an empty set $visited$
Initialize an empty dictionary $cell\_boundaries$

procedure DFSTraversalDist($CB$)
    Compute $CB$
    Add $CB$ to $visited$
    Insert $CB$ in $cell\_boundaries$

    for each $neighbor$ of $CB$ in $nodeLink$ do
        if $neighbor$ is not in $visited$ then
            DFSTraversalDist($CB$)
        end if
    end for
end procedure

$seed\_CB = cellBoundary(e_1, v_1)$                    ▷ Starting cell boundary.
Call DFSTraversalDist($seed\_CB$)
```

Step 2: Verify the Traversal of G_2. Following the traversal detailed in **Step 1**, it is necessary to next verify that G_2 was entirely traversed. As previously discussed, this verification `projection_check` is determined true if the projection of `cell_boundaries` onto G_2 covers the entirety of the graph and is determined false otherwise. Furthermore, it is implied that $\delta_T(G_1, G_2) \leq \epsilon$ when `projection_check` equals true and $\delta_T(G_1, G_2) > \epsilon$ when `projection_check` equals false. Let the output of `projection_check` be:

$$projection\_check(cell\_boundaries, G_2) = \begin{cases} \text{True} & \text{Projection covers } G_2 \text{ entirely.} \\ \text{False} & \text{Otherwise.} \end{cases}$$

Step 3: Binary Search for Traversal Distance. The algorithms established in **Step 1** and **Step 2** will now be incorporated into the binary search algorithm, denoted as `binarySearch`. Defined in Algorithm 2, this particular algorithm approximates the infimum of the traversal distance equation. It achieves this by searching for the smallest value of ϵ, for which `projection_check` yields true.

Algorithm 2 binarySearch

```
procedure BINARYSEARCH($left$, $right$, $precision$)
    Initialize $\epsilon$

    while $right - left > precision$ do
        $\epsilon \leftarrow (left + right)/2$
        Initialize $cell\_boundaries$
        Call $DFSTraversalDist(CB)$

        if $projection\_check(cell\_boundaries, G_2)$ is True then
            $right \leftarrow \epsilon$
        else
            $left \leftarrow \epsilon$

        end if
    end while
    return $right$
end procedure
```

Given the continuous domain of the search space for ϵ, since $\delta_T(G_1, G_2) \in [0, \infty)$, it is necessary to bound ϵ within a finite search domain. Thus, we assume $\delta_T(G_1, G_2) \in [left, right]$, ensuring that the `binarySearch` reduces the space to $[\epsilon - precision, \epsilon]$. Here, `left` and `right`, respectively, denote the lower and upper boundaries of the search space, while `precision` specifies the degree of accuracy to which the value of ϵ is returned.

Having explained the steps of the traversal distance algorithm, it is now evident that the traversal distance is computed in the Python program by calling the `binarySearch` function.

2.5 Properties of the Traversal Distance

Space Complexity of the Cell Boundaries After running the traversal distance algorithm, the program stores the values of ϵ and `cell_boundaries`. To determine the program memory requirements for this, we calculate the upper bound of the number of cell boundaries that could exist between two geometric graphs. This calculation reveals that the space complexity of `cell_boundaries`, given two geometric graphs G_1 and G_2, is asymptotically bound by [13]:

$$S \in O((|V_1| \cdot |E_2|) + (|V_2| \cdot |E_1|))$$

Time Complexity of the Traversal Distance The time complexity of the traversal distance algorithm is determined by the combined time complexities of the `DFSTraversalDist`, `projection_check`, and `binarySearch` algorithms.

First, consider the fact that `DFSTraversalDist` is, by definition, a DFS algorithm, which runs in polynomial time. If we assume the time is takes to compute *cellBoundry* $\in O(1)$ then the time complexity of `DFSTraversalDist` is:

$$T_D \in O((|V_1| + |E_1|) \cdot (|V_2| + |E_2|))$$

The `projection_check` runs in polynomial time such that [1]:

$$T_P \in O((|V_1| \cdot |E_2|) + (|V_2| \cdot |E_1|))$$

For `binarySearch` operating over a continuous space, its time complexity is influenced by the number of iterations required to achieve the desired precision within the defined bounds. Given the `left` bound, `right` bound, and `precision`, the time complexity can be articulated as:

$$T_B \in O(\log_2 \Delta) \quad \text{where} \quad \Delta = \frac{right - left}{precision}$$

Combining these individual time complexities, we can determine the cumulative time complexity of the traversal distance algorithm as follows:

$$\begin{aligned}
T_T &= T_B \cdot (T_D + T_P) \\
&= T_B \cdot O((|V_1| + |E_1|) \cdot (|V_2| + |E_2|)) + O((|V_1| \cdot |E_2|) + (|V_2| \cdot |E_1|)) \\
&= T_B \cdot O(|V_1||E_2| + |V_2||E_1| + |V_1||E_1| + |V_2||E_2|) \\
&= O(\log_2 \Delta) \cdot O(|V_1||E_2| + |V_2||E_1| + |V_1||E_1| + |V_2||E_2|) \\
&= O(\log_2 \Delta \cdot (|V_1||E_2| + |V_2||E_1| + |V_1||E_1| + |V_2||E_2|))
\end{aligned}$$

This analysis establishes that the algorithm runs in $O(\log_2 \Delta \times (|V_1||E_2| + |V_2||E_1| + |V_1||E_1| + |V_2||E_2|))$ time [13]. Another property to revisit, as previously mentioned, is the traversal distance as an asymmetric measure. For the remainder of this thesis, it is asserted that [1]:

$$\delta_T(G, G) = 0 \quad \forall G$$

2.6 Geometric Graph Edit Distance

A second method for measuring the distance between two geometric graphs involves calculating the edit distance between them. The edit distance between two objects is defined as the minimum number of operations required to transform one object into the other, where the operations include insertions, deletions, and relabeling [7]. Similar to how the Levenshtein distance measures the edit distance between two strings [8], the edit distance between two geometric graphs is referred to as the graph edit distance (GED).

For geometric graphs G_1 and G_2, GED searches for a sequence of operations p that transforms $G_1 \to G_2$ such that the transformed graph $G_1' = G_2$. An operation o may include deleting isolated vertices, inserting vertices, adding edges between existing vertices, deleting edges, and translating a vertex from one point to another. Each operation o has a corresponding cost function $Cost(o)$ for executing the operation. The cost for a sequence of operations is given by:

$$Cost(p) = \sum_{o_i \in p} Cost(o_i) \quad \text{where} \quad p := \text{sequence of operations}$$

As a result, GED is defined as the cost of the least expensive path that transforms $G_1 \to G_2$.

$$GED(G_1, G_2) = \inf_{p \in P(G_1, G_2)} Cost(p)$$

$$P(G_1, G_2) := \text{Set of all } p \text{ that transform } G_1 \to G_2$$

An advantage of GED over the traversal distance is its symmetry, which exists since the edit sequences between geometric graphs are reversible. One significant disadvantage is that computing GED is NP-hard, meaning that the time required to compute GED increases exponentially with the size of the geometric graphs [7].

3 Visualizing the Traversal Distance

This chapter builds on the traversal distance by demonstrating two visualization techniques. It starts with the free-space diagram for the weak Fréchet distance, explaining how it facilitates the reader's understanding of the concept. Additionally, a new visualization method for the traversal distance is introduced, addressing the challenges posed by the idea of a traversal distance free-space diagram.

3.1 Weak Fréchet Distance Free-Space Diagram

An important tool for visualizing the weak Fréchet distance is the FS_ϵ diagram. This visualization plays a role in enhancing the understanding of what the weak Fréchet distance algorithm computes. It also allows for the verification of whether cell boundaries within the diagram are being computed correctly [1]. The following discussion delves into the construction and interpretation of the FS_ϵ diagram, and how it builds on the visualization of the traversal distance free-space. The visualization in this section were generated using the Fréchet distance Python program documented in the appendix.

In Figures 8, 9 and 10, the FS_ϵ diagram is demonstrated through a simple example involving two curves, C_1 and C_2. In Figure 8, Curve C_1 is comprised of three line segments, while C_2 is comprised of four line segments. In Figures 9 and 10, Curve C_1 is aligned along the horizontal axis and C_2 along the vertical axis, with FS_ϵ represented in white.

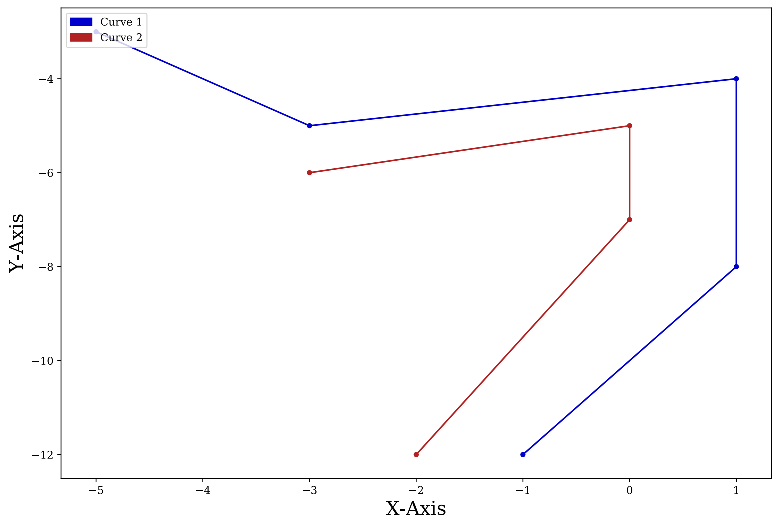


Figure 8: Curves C_1 and C_2 [13].

Assigning an arbitrary epsilon value, $\epsilon = 2$, to the FS_2 diagram, the resulting visualization of the diagram can be observed as follows.

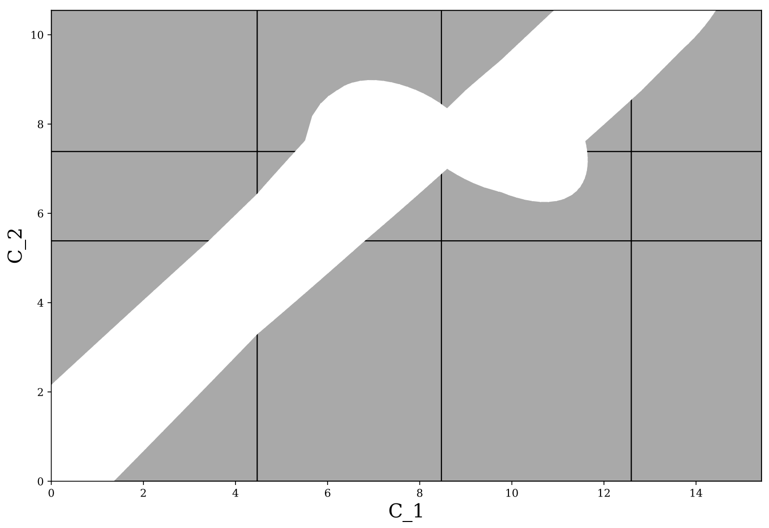


Figure 9: Free-space diagram where $\epsilon = 2$ [13].

Observing Figure 8, the diagram consists of 12 free-space cells, matching the product of line segments in each curve: four in C_1 and three in C_2. This epsilon value is not considered a weak Fréchet distance, however, since FS_2 does not cover every line segment inside C_1.

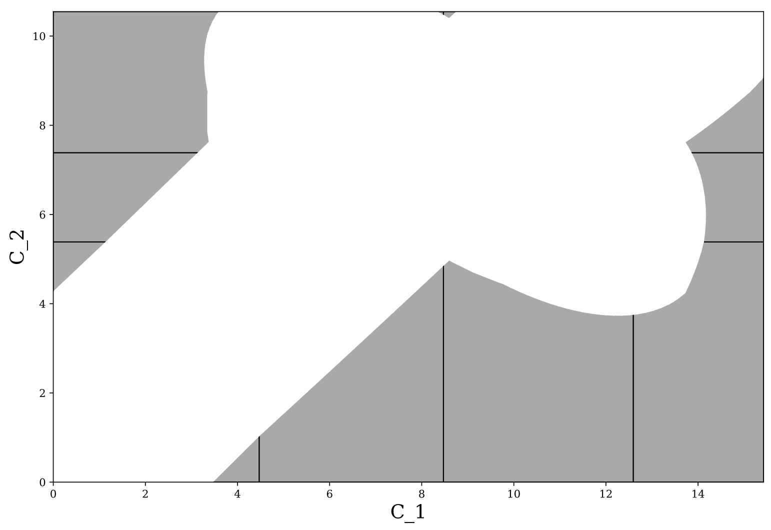


Figure 10: Free-space diagram where $\epsilon = 4$ [13].

Increasing epsilon to $\epsilon = 4$ in Figure 10 allows FS_4 to cover every line segment inside C_1 and C_2, indicating that $\delta_F(C_1, C_2) \leq 4$ [13].

3.2 Traversal Distance Free-Space Visualization

Shifting focus to the traversal distance, a problem arises when visualzing the FS_ϵ diagram. The FS_ϵ diagram for the weak Fréchet distance can be effectively displayed in the Euclidean plane; this is achieved by mapping C_1 and C_2 along the X and Y axes in the Euclidean plane, respectively. However, this approach encounters a limitation when applied to geometric graphs. Such graphs cannot be similarly reduced to an axis in the Euclidean plane without causing free-space cells to overlap.

To understand how geometric graphs cause free-space cells to overlap, consider the case of a free-space diagram for one polygonal curve C and one geometric graph G. Figure 11 shows an example of C and G.

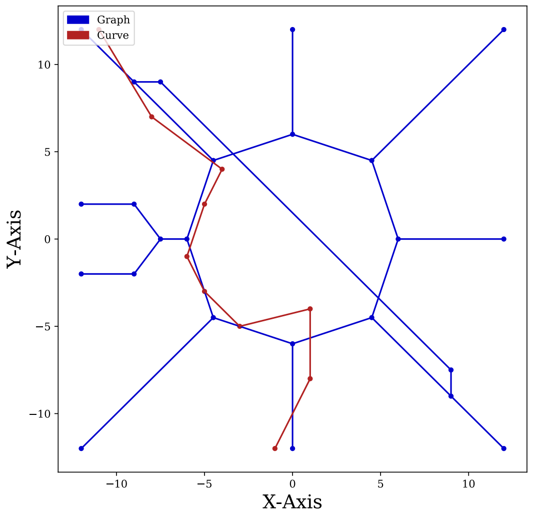


Figure 11: Curve C and graph G. [13].

The free-space for Figure 11 can be visualized by constructing a free-space diagram of 2D planes in 3D space, where G is on the X and Y plane, and C is aligned along the Z axis. Figure 12 visualizes the free-space diagram of G and C with $\epsilon = 5$, as seen from an arbitrary viewpoint.

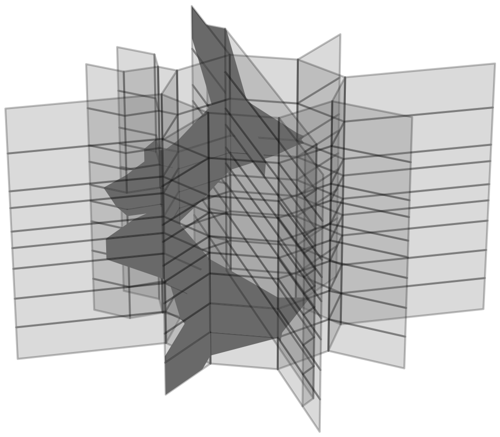


Figure 12: Free-space diagram of G and C where $\epsilon = 5$ [13]

From Figure 12, it is evident that there is no viewpoint of the free-space diagram where the free-space cells do not overlap. This same challenge exists for the case of a traversal distance free-space diagram, where a free-space diagram is constructed from 2D planes in 4D space. Consequently, 4D space cannot be directly visualized. This section introduces a method for visualizing the traversal distance's free-space. Instead of displaying a free-space diagram with cells, this method focuses on visualizing the area of the free-space within the free-space cells.

Area of Free-Space Within a Cell In the case of two edges e_i and e_j, each from geometric graphs G_1 and G_2, these two edges constitute a single free-space cell, labeled $FS_{\epsilon,i,j}$. Let A be the area of the free-space polygon inside the free-space cell $FS_{\epsilon,i,j}$, denoted $A(FS_{\epsilon,i,j})$. Recall that $FS_{\epsilon,i,j}$ is represented as a square cell, constructed with four boundary walls, and the length from `start` to `end` of a cell boundary falls within the range $[0, 1]$. Since a free-space cell is a square of length one, then $A(FS_{\epsilon,i,j}) \in [0, 1]$.

Since A is a value between $[0, 1]$, it can be interpreted as a percentage. For instance, if $A(FS_{\epsilon,i,j}) = 0.345$, this indicates that 34.5% of $FS_{\epsilon,i,j}$ is covered by free-space. $A(FS_{\epsilon,i,j}) = 0.0$ would indicate $FS_{\epsilon,i,j}$ is entirely grey and $A(FS_{\epsilon,i,j}) = 1.0$ would indicate it is entirely white.

Visualizing Free-Space Area To visualize the free-space area within $FS_{\epsilon,i,j}$, color in the quadrilateral area that lies between both edges e_i and e_j on their Euclidean plane. Consider this quadrilateral area the spatial relationship between these two edges. Figure 13 demonstrates the spatial relationship between an example pair of edges e_i and e_j.

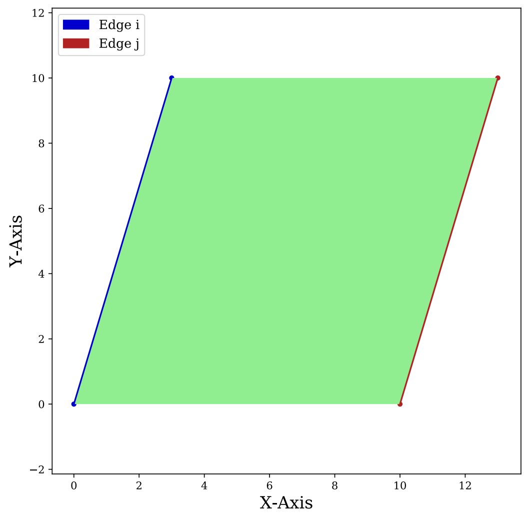


Figure 13: Highlighted space between e_{1_i} and e_{2_j} [13].

Furthermore, a transparency function α is applied to the color of the quadrilateral area. Such that the degree of transparency directly corresponds to the value of $A(FS_{\epsilon,i,j})$. This means that α visually represents the proportion of free-space covering cells. Figures 14 through 17 illustrate how, as ϵ increases from an empty free-space cell when $\epsilon = 0$ to a full free-space cell when $\epsilon = 15$, the color's transparency decreases.

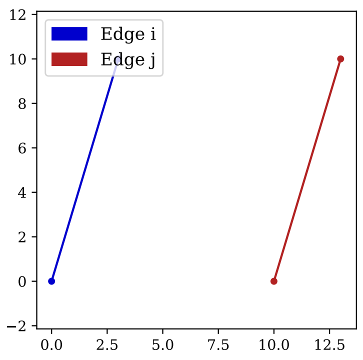


Figure 14: $\epsilon = 0$ [13].

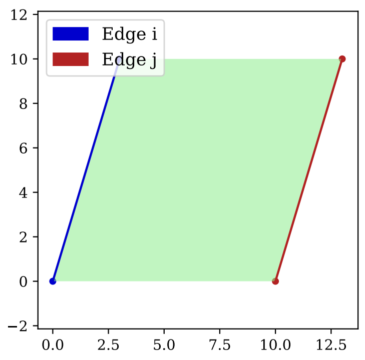


Figure 15: $\epsilon = 10.01$ [13].

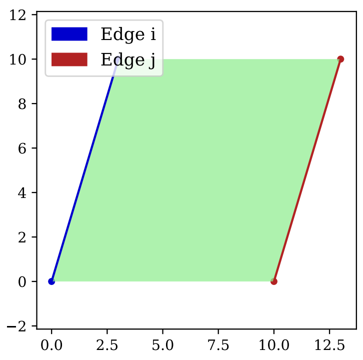


Figure 16: $\epsilon = 10.5$ [13].

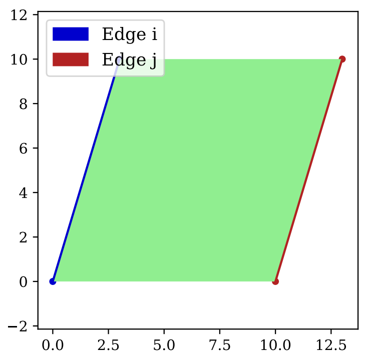


Figure 17: $\epsilon = 15$ [13].

This value α is amplified for overlapping areas. Consider a set of overlapping free-space areas $S = \{A_1, A_2, \ldots, A_n\}$. The value of α for the intersection of these overlapping areas is calculated as follows [13]:

$$\alpha = 1 - \prod_{i=1}^{n}(1 - A_i)$$

3.3 Example of Traversal Distance Visualization

To demonstrate the overlapping property, consider an example involving a pair of geometric graphs representing more complex structures. This example involves a comparison between two distinct species of plant leaves, with each leaf constructed as a geometric graph. In these graphs, the edges represent both the outline of the leaf and its vein structure.

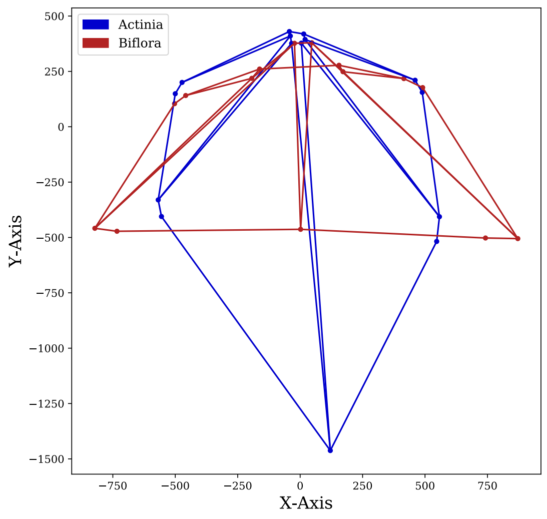


Figure 18: Two plant leaves, from the species Actinia and Biflora, as geometric graphs [11].

Having plotted the pair of geometric graphs, the free-space between both geometric graphs is now colored in for several values of ϵ.

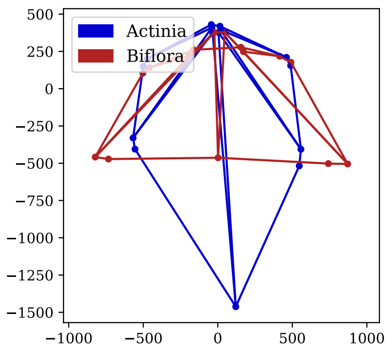


Figure 19: $\epsilon = 0$ [11].

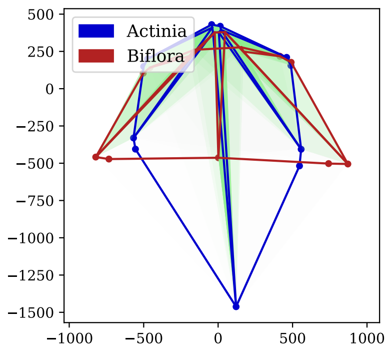


Figure 20: $\epsilon = 150$ [11].

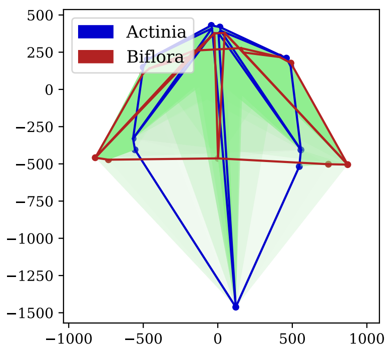


Figure 21: $\epsilon = 300$ [11].

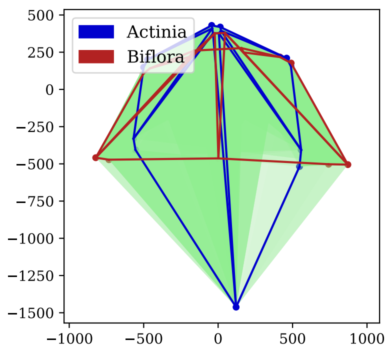


Figure 22: $\epsilon = 450$ [11].

Observing the visualizations, it becomes evident that the highlighted free-space expands as the value of epsilon increases [11].

4 Distance Measurements in Machine Learning

This chapter switches focus to concepts in machine learning: non-parametric supervised learning and classification. These topics will be important for differentiating different types of models within the field of machine learning. Followed by the application of distance measures in machine learning, with a particular focus on the k-nearest neighbors (k-NN) model. The chapter introduces the concept of generalizing distance, explaining how k-NN, as defined here, performs when used with the traversal distance in chapter 5.

4.1 Introducing the K-Nearest Neighbors Model

k-NN is a classification model in machine learning, specifically within supervised learning. As a non-parametric model, k-NN distinguishes itself by assuming the data does not have any specific underlying statistical distribution. This characteristic of being assumption-free renders k-NN not only basic but also an essential model for grasping the concepts in machine learning classification.

Classification in Supervised Learning Supervised learning in machine learning refers to the process where a model is trained using a labeled dataset. Where the term supervised implies that the model learns from the input data X and output label y. In other words, the supervision of learning can be thought of as a function $model(X) = y$.

Classification in supervised learning involves categorizing data into predefined classes. A model predicts the class of a new observation x_{new} by analyzing X, then assigns x_{new} a predicted label $\hat{y}$.

Classification can be separated into two formats: binary and multi-class. In binary classification, the model classifies observations into one of two classes. While multi-class classification involves categorizing observations into one of multiple classes. This thesis focuses only on multi-class classification [5].

K-Nearest Neighbors Algorithm The process of predicting a class, for an observation x_{new}, can be broken down into the following steps:

1. Choose the Number of Neighbors: Select the number of nearest neighbors, k, which will influence the prediction. How to determine a value for k is discussed in the appendix.

2. Calculate Distances: For the observation x_{new}, compute the Euclidean distance between x_{new} and each observation in X.

3. Identify Nearest Neighbors: Sort all calculated distances in ascending order. Then select the top k nearest observations from the dataset.

5. Predict the Classification: Determine the predicted label $\hat{y}$ for the observation x_{new} by taking the most common label among the k neighbors. This step is know as a majority voting ensemble in machine learning.

The formal definition of the k-NN algorithm is presented as Algorithm 3 in [10].

Algorithm 3 KNearestClassifier

```
Require: Dataset $X$
Require: Labels $y$
Require: New observation $x_{new}$

  Initialize number of neighbors $k$
  Initialize distance $d$
  Initialize list *distances*

  Begin Algorithm
  for each point $x_i$ in the training data $X$ do
      Calculate distance $d(x_i, x_{\text{new}})$
      Append $d(x_i, x_{\text{new}})$ to *distances*
  end for

  Sort *distances* in ascending order
  Select top $k$ nearest neighbors from *distances*
  Aggregate labels of selected neighbors
  $\hat{y}$ := Most common label
  return $\hat{y}$
  End Algorithm
```

A sixth step of the algorithm, not previously discussed, addresses situations where majority voting leads to a tie. This occurs when two or more labels are equally most common among the aggregated labels [5].

Time Complexity If the time complexity of our traversal distance algorithm is T_T. Assuming the sorting algorithm used is Quick Sort, the time complexity of the sorting process can be defined as $O(|X| \log_2 |X|)$ [3], where $|X|$ represents the number of observations in the data. Therefore, the overall time complexity of a k-NN prediction can be expressed as follows [5]:

$$T_{KNN} \in O(T_T|X| + |X| \log_2 |X|)$$

4.2 Evaluating K-Nearest Neighbors Predictions

This section outlines how to assess the performance of k-Nearest Neighbors (k-NN) models using n-fold cross-validation, alongside metrics such as precision and recall. Explaining n-fold cross-validation, which involves dividing the dataset into multiple parts to ensure each segment is used for testing. This approach helps measure the model's ability to predict new observations, as well as the comparison of different model's. The application of these methods will be demonstrated using the scikit-learn library [10].

N-Fold Cross-Validation To evaluate the k-NN algorithm, we use n-fold cross-validation, where the dataset is split into n equal parts, called folds. In each evaluation round, one fold is used as the test set and the other $n-1$ folds as the reference set. This cycle is repeated until each fold has been used as the test set once, ensuring all data is used for both training and testing. Below is how n-fold cross-validation is implemented using scikit-learn:

```
from sklearn.model_selection import KFold

folds = KFold(n_splits = 5)
```

Precision and Recall of Predictions To measure how many predicted values ($\hat{y}$) in a test fold are correctly classified as the actual values (y), precision and recall metrics are used. Precision measures the proportion of correctly predicted positive observations to the total predicted positive observations. It is a key indicator of a model's ability to minimize false positives. Recall, on the other hand, assesses the proportion of actual positive observations that were correctly predicted by the model, thus reflecting its capability to minimize false negatives. For multi-classification problem, the Precision and Recall of a set of predictions is defined as [4]:

Definition 5. *Given a set of multiple classes $C = \{c_i | i \in \mathbb{N}\}$, within a set of predictions.*

- ***Precision*** *for a specific class c_i is defined as the ratio of the number of true positive instances TP_i to the total number of instances predicted as belonging to that class, which is the sum of the true positives and false positives FP_i:*

$$P_i = \frac{TP_i}{TP_i + FP_i}$$

- ***Recall*** *for class c_i is the ratio of the number of true positive instances TP_i to the actual number of instances of that class in the data, which is the sum of the true positives and false negatives FN_i:*

$$R_i = \frac{TP_i}{TP_i + FN_i}$$

The calculation of precision and recall can be performed using scikit-learn as follows [10]:

```
from sklearn.model_selection import cross_val_score

p = cross_val_score(model, X, y, cv=folds, scoring='precision')
r = cross_val_score(model, X, y, cv=folds, scoring='recall')
```

5 Applying the Traversal Distance to Classification Problems

To test the precision of the traversal distance within the framework of the k-nearest neighbors (k-NN) algorithm, the inherent asymmetry of the traversal distance must be addressed. Therefore, a symmetric adaptation suitable for k-NN is proposed. This adjustment enables the k-NN algorithm to classify a dataset of English letters represented as geometric graphs using traversal distance [12]. The goal is to evaluate the hypothesis that traversal distance can precisely classify geometric graphs. This is determined by comparing the performance of the traversal distance k-NN model with that of the graph edit distance (GED) k-NN model.

5.1 Symmetric Case of the Traversal Distance

To incorporate the traversal distance into the k-NN algorithm, it is necessary to first address the asymmetry of the traversal distance. Meaning, the distance from G_1 to G_2 may not equal the distance from G_2 to G_1, resulting in two distinct distances. k-NN operates under the assumption that the distance metric implemented is symmetric [5]. Consequently, a symmetric variant of the traversal distance must be defined.

To develop a symmetric distance metric for k-NN, a function must be devised that combines the two asymmetric distances produced by the traversal distance into a single distance measure [1]. The design of a function should be tailored to the specific requirements of a dataset [5]. Given that this chapter concentrates on the comparison of English characters, it operates under the presumption that the geometric graphs being compared have equivalent magnitudes [12].

For the distance metric of k-NN, the function will be defined as the maximum of the two distances produced by the traversal distance. Taking the maximum value ensures that the distance, denoted by ϵ, completely covers both geometric graphs during `projection_check`.

Theorem 1. *Let the symmetric traversal distance between two geometric graphs G_1 and G_2 be defined by the equation:*

$$\delta_{ST}(G_1, G_2) = \max\{\delta_T(G_1, G_2), \delta_T(G_2, G_1)\}$$

Then, it holds that $\epsilon = \delta_{ST}(G_1, G_2)$ passes the `projection_check` for both $\delta_T(G_1, G_2)$ and $\delta_T(G_2, G_1)$, ensuring that ϵ fully covers both G_1 and G_2 for the symmetric case.

Proof. For the symmetric traversal distance between two geometric graphs G_1 and G_2, assume without loss of generality:

$$\delta_{ST}(G_1, G_2) = \delta_T(G_1, G_2) \geq \delta_T(G_2, G_1)$$

Where $\delta_T(G_1, G_2) = \epsilon_1$ and $\delta_T(G_2, G_1) = \epsilon_2$. This implies that $\epsilon_1 \geq \epsilon_2$. By **Definition 4**, $\delta_T(G_1, G_2)$ traverses entirely over G_1 and traverses partially over G_2. Given that the free-space is a monotone function of ϵ, this statement holds for any $\epsilon \in [\epsilon_2, \infty)$. If $\epsilon_1 \geq \epsilon_2$, then $\epsilon_1 \in [\epsilon_2, \infty)$, ensuring $\delta_T(G_1, G_2)$ traverses over the entirety of both G_1 and G_2. Therefore, $\delta_{ST}(G_1, G_2)$ covers both graphs for the purpose of `projection_check`. $\square$

5.2 K-Nearest Neighbors Using the Traversal Distance

Having established a symmetric traversal distance, it is now possible to predict classifications using the English letter dataset. In the context of the English Letter dataset, X comprises of English letters stored as geometric graphs rather than traditional points, y denotes the class of letter, and d serves as the metric for quantifying the distance between any two geometric graphs. The dataset comprises 2,250 labeled geometric graphs, each representing a distorted drawing of an English letter. These drawings are categorized into 15 distinct classes of 150 observations. Each class corresponds to one letter such that $C = \{A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z\}$ [12]. See Figures 23 though 26 for examples of geometric graphs.

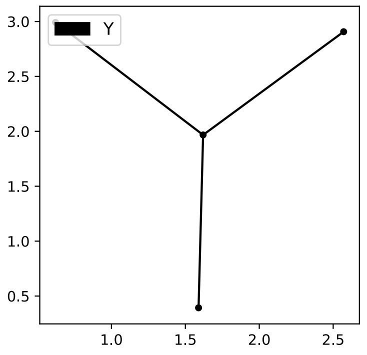


Figure 23: Class Y [12].

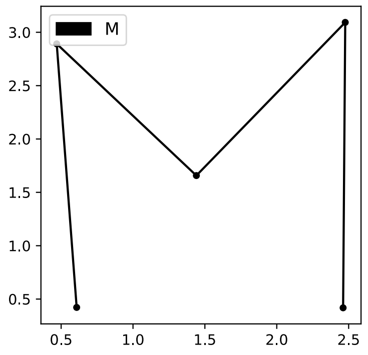


Figure 24: Class M [12].

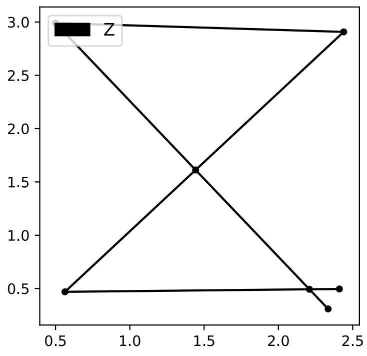


Figure 25: Class Z [12].

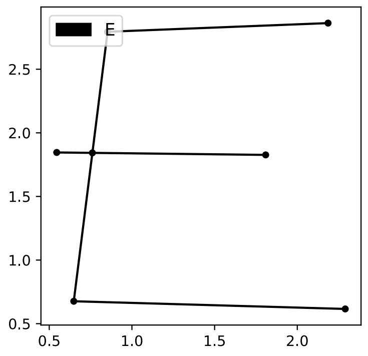


Figure 26: Class E [12].

To perform n-fold cross-validation, the dataset is partitioned into $n = 5$ folds using uniform random sampling [5]. For k-NN, the parameter specifying the number of neighbors, is set to $k = 7$. Meanwhile, for the traversal distance calculation, the `binarySearch` function's parameters are configured as follows: $left = 0$, $right = 3$, and $precision = 0.001$.

```
model = KNeighborsClassifier(n_neighbors=7, mean='max',
                             left=0, right=3, precision=0.001)
```

The model is first fitted on the training dataset, then predicts the observations in the test dataset, yielding 449 predicted classes from the test dataset, denoted as $\hat{y}$ [5]. To evaluate the precision and recall of the $|C| = 15$ classes between $\hat{y}$ and the true labels y, both macro-average precision and macro-average recall are calculated. These metrics are computed by first calculating the precision for each class independently then taking the average of these precision scores [4].

$$\text{Macro-Average Precision} = \frac{1}{|C|}\sum_{i=1}^{|C|} P_i \quad \text{Macro-Average Recall} = \frac{1}{|C|}\sum_{i=1}^{|C|} R_i \tag{1}$$

After running the test, the results showed the macro-average precision's and a macro-average recall's listed in Table 1.

Test Fold	**Macro-Average Precision**	**Macro-Average Recall**
1	88.0	87.6
2	86.3	85.1
3	87.9	88.0
4	84.6	83.5
5	81.6	82.5

Table 1: Macro-average precision and recall scores.

By taking the averages of these five macro-average precision's and macro-average recalls, the average values are 85.7 percent and 85.3 percent respectively, below those obtained when using k-NN with the graph edit distance [12].

5.3 K-Nearest Neighbors Using Graph Edit Distance

To compare the results obtained using traversal distance, it is useful to examine a benchmark test conducted with GED. In their seminal work, Kaspar Riesen and Horst Bunke investigate the efficacy of the k-NN model, employing GED, in classifying geometric graph representations of the English alphabet. This comparison serves as the reference point for assessing the performance of traversal distance.

In their investigation, Riesen and Bunke utilize a dataset comprising 6,750 observations of English letters, with 750 observations uniformly distributed across 15 letter classes. To evaluate the performance of the GED k-NN model, the dataset was divided into equal parts for validation, training, and testing. As a result, the model achieved a macro-average precision of 99.6 percent on the test dataset, showcasing the high performance of GED in this context [12].

When compared, the GED k-NN model outperforms the traversal distance k-NN model by 16.2 percent.

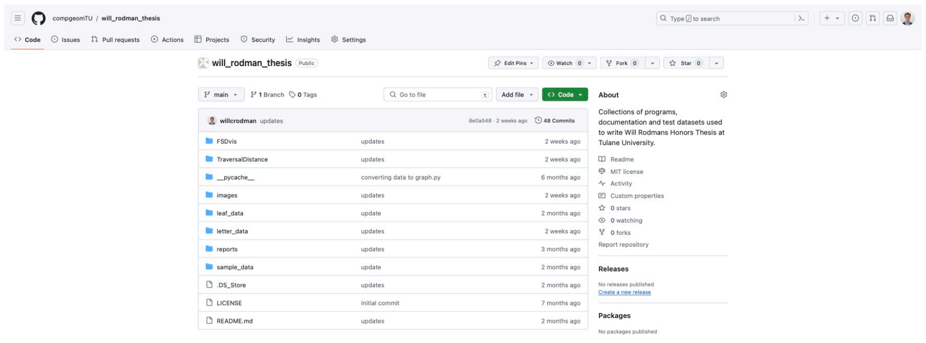
6 Conclusion

This thesis examined the application of geometric graph distances within computer science, with a specific focus on the traversal distance and the algorithm for computing it. The aim was to assess the utility of the traversal distance in classifying geometric graphs within the English letter dataset. Comparative analysis revealed that the k-NN model based on traversal distance achieved precision comparable to that of the GED-based k-NN model by Riesen and Bunke, despite the traversal distance being polynomial and GED being NP-Hard. This result indicates that the traversal distance is an effective method for classifying geometric graphs.

The research presented several key findings. The traversal distance was defined by incorporating elements of geometric graph theory, weak Fréchet distance, and free-space diagrams. The algorithm's steps were detailed, and its time complexity was determined to be polynomial. Additionally, the thesis proposed a method for visualizing free-space areas related to the traversal distance, demonstrating its effectiveness with examples from the plant leaf dataset. A symmetric version of the traversal distance was defined, wherein taking the maximum value of both asymmetric distances ensures that the free-space entirely covers both geometric graphs. The efficacy of the traversal distance in enhancing the k-NN model's performance for classifying geometric graphs, as evidenced in the English letter dataset, was measured with precision and recall metrics.

For future work, the thesis suggests avenues for refining the traversal distance. The symmetric traversal distance algorithm could be optimized by preventing the recalculation of free spaces already computed within the maximum function. Moreover, there is potential for enhancing the overall runtime efficiency of the traversal distance algorithm through the adoption of a lower-level programming language. In conclusion, the thesis underscores that distances in geometric graph theory represent a relatively new field within mathematics that is currently evolving. The application of these distances, particularly in computer science for digital road maps and increasingly within supervised machine learning, is expected to broaden as the efficiency of these algorithms improves and their comparative advantages are tested against existing software and models.

Appendix



will_rodman_thesis Public

1 Branch 0 Tags

willcrodman updates 8e0a548 · 2 weeks ago 48 Commits

FSDvis	updates	2 weeks ago
TraversalDistance	updates	2 weeks ago
__pycache__	converting data to graph.py	6 months ago
images	updates	2 weeks ago
leaf_data	update	2 months ago
letter_data	updates	2 weeks ago
reports	updates	3 months ago
sample_data	update	2 months ago
.DS_Store	updates	2 months ago
LICENSE	Initial commit	7 months ago
README.md	updates	2 months ago

About

Collections of programs, documentation and test datasets used to write Will Rodmans Honors Thesis at Tulane University.

Readme
MIT license
Activity
Custom properties
0 stars
0 watching
0 forks
Report repository

Releases

No releases published
Create a new release

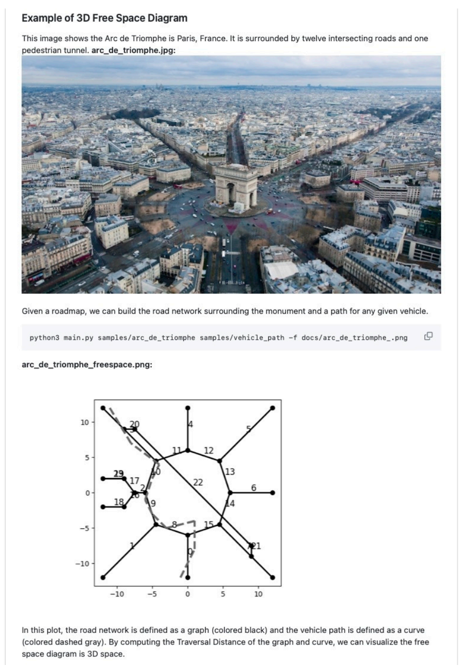
Packages

No packages published

Figure 27: Image of honors thesis GitHub repository [13].

This appendix documents the Python packages and Jupyter Notebooks written to support this thesis. All supporting materials are publicly available in the `compgeomTU/will_rodman_thesis` GitHub repository.

FSDVis Python Package



Example of 3D Free Space Diagram

This image shows the Arc de Triomphe is Paris, France. It is surrounded by twelve intersecting roads and one pedestrian tunnel. **arc_de_triomphe.jpg:**

Given a roadmap, we can build the road network surrounding the monument and a path for any given vehicle.

```
python3 main.py samples/arc_de_triomphe samples/vehicle_path -f docs/arc_de_triomphe_.png
```

arc_de_triomphe_freespace.png:

In this plot, the road network is defined as a graph (colored black) and the vehicle path is defined as a curve (colored dashed gray). By computing the Traversal Distance of the graph and curve, we can visualize the free space diagram is 3D space.

Figure 28: Image of `will_rodman_thesis/FSDVis` ReadMe file [13].

The FSDVis Python Package computes and visualizes the free-space diagram for the weak Fréchet distance, as well as the free-space diagram for a single geometric graph and a polygonal curve. The package is located at `will_rodman_thesis/FSDVis`. Using Figure 12 as an example, the following script from Jupyter Notebook `image_generator.ipynb` imports the package and plots a free-space diagram.

```
import matplotlib.pyplot as plt
from FSDVis.Curve import Curve
from FSDVis.Graph import Graph
from FSDVis.GraphByCurve import GraphByCurve

filepath_1 = 'examples/paris/arc_de_triomphe'
filepath_2 = 'examples/paris/vehicle'
graph = Graph(filepath_1)
curve = Curve(filepath_2)
epsilon = 5

fsd = GraphByCurve(graph, curve)
fsd.buildFreeSpace(epsilon)
fsd.buildCells()
fsd.plotFreeSpace()
```

TraversalDistance Python Package

The TraversalDistance Python Package contains the traversal distance algorithm, traversal distance visualizer and k-NN algorithm implementing the symmetric traversal distance. The package located at `will_rodman_thesis/TraversalDistance` contains the files listed in Table 2.

File	**Description**
BinarySearch.py	`BinarySearch` class function of the traversal distance algorithm.
CalFreeSpace.py	Function for computing then returning the $[start, end]$ boundary for free-space cell walls.
FreeSpaceGraph.py	`DFSTraversalDistance` class function of the traversal distance algorithm.
Graph.py	Class data structure for storing geometric graphs.
KNeighborsClassifier.py	Custom k-NN class that implements the symmetric traversal distance.
LineIntersection.py	Supporting class for the `projection_check` function.
Visualize.py	Class responsible for visualizing the traversal distance free-space area.

Table 2: TraversalDistance Python Package Files.

Using Figure 20 as an example, the following script from Juniper Notebook `image_generator.ipynb` imports the package and plots the traversal distance free-space area.

```
import matplotlib.pyplot as plt
from TraversalDistance.Graph import Graph
from TraversalDistance.Visualize import Visualize

filepath_1 = 'examples/plant/Pact1_actinia_3'
filepath_2 = 'examples/plant/Pbif1_biflora_1'
graph_1 = Graph(filename=filepath_1, name='Actinia')
graph_2 = Graph(filename=filepath_2, name='Biflora')

visualize = Visualize(graph_1, graph_2, epsilon=150)

fig, ax = visualize.plot_freespace(legend_fontsize='large')
fig.set_size_inches(4, 4)
plt.show()
```

Traversal Distance K-NN Model Test: Jupyter Notebooks

The analysis of the traversal distance k-NN model, conducted on the English letter dataset, was executed within the Python script `letter_knn.py`. To predict all the values in the dataset, the model was run on Google Could Platforms Compute Engine. Test results were logged and subsequently analyzed in the Jupyter notebook `letter_knn_analysis.ipynb`. This analysis, illustrated in Figure 29, demonstrates how precision and recall metrics for an n-fold vary in relation to the value of k.

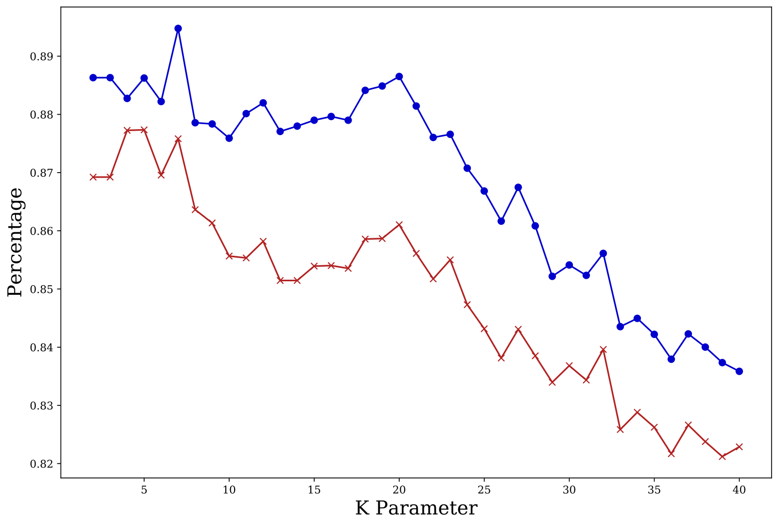


Figure 29: Traversal distance k-NN model precision and recall scores on the English letter dataset

Upon reviewing Figure 29, both precision and recall exhibit a consistent decline as k increases from 20 to 40, with peak precision occurring at $k = 7$.

References

[1] Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003.

[2] Daniel Chen, Christian Sommer, and Daniel Wolleb. Fast Map Matching with Vertex-Monotone Fréchet Distance. In Matthias Müller-Hannemann and Federico Perea, editors, *21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021)*, volume 96 of *Open Access Series in Informatics (OASIcs)*, pages 10:1–10:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[3] Shalosh B. Ekhad and Doron Zeilberger. A Detailed Analysis of Quicksort Running Time. https://arxiv.org/abs/1903.03708, 2019.

[4] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for Multi-Class Classification: an Overview. https://arxiv.org/abs/2008.05756, 2020.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2 edition, 2009.

[6] Erfan Hosseini. GraphSamplingToolkit: Compare roadmaps or evaluate reconstructed maps with Graph Sampling Toolkit. `https://github.com/Erfanh1995/GraphSamplingToolkit`, 2021.

[7] Sushovan Majhi and Carola Wenk. Distance Measures for Geometric Graphs. https://arxiv.org/abs/2209.12869, 2022.

[8] Evgenii Ofitserov, Vasily Tsvetkov, and Vadim Nazarov. Soft edit distance for differentiable comparison of symbolic sequences. https://arxiv.org/pdf/1904.12562.pdf, 2019.

[9] Janos Pach. Geometric Graph Theory, 2017. `https://api.semanticscholar.org/CorpusID:46340637`.

[10] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. *Scikit-learn: Machine Learning in Python.* Scikit-learn developers, 2023.

[11] Sarah Percival, Joyce Onyenedum, Daniel Chitwood, and Aman Husbands. Topological data analysis reveals core heteroblastic and ontogenetic programs embedded in leaves of grapevine (Vitaceae) and maracuyá (Passifloraceae). *PLoS computational biology*, 20:e1011845, 02 2024.

[12] Kaspar Riesen and Horst Bunke. IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning. In Niels da Vitoria Lobo, Takis Kasparis, Fabio Roli, James T. Kwok, Michael Georgiopoulos, Georgios C. Anagnostopoulos, and Marco Loog, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 287–297, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[13] Will Rodman. will_rodman_thesis: Traversal Distance Python Library. `https://github.com/compgeomTU/will\_rodman\_thesis`, 2024.

[14] Carola Wenk. Weak Fréchet code. `https://www.cs.tulane.edu/~carola/research/code.html`, 2018.