2012

Experimental Approaches to Computational Geometric and Statistical Machine Translation Problems

Samidh Chatterjee
EXPERIMENTAL APPROACHES TO COMPUTATIONAL GEOMETRIC AND
STATISTICAL MACHINE TRANSLATION PROBLEMS

By

SAMIDH CHATTERJEE

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Degree Awarded:
Fall Semester, 2012
Samidh Chatterjee defended this dissertation on October 24, 2012.
The members of the supervisory committee were:

Piyush Kumar
Professor Directing Dissertation

Gary Tyson
Committee Member

Michael Mascagni
Committee Member

Washington Mio
University Representative

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with the university requirements.
Dedicated to my parents, Mrs. Tapati Chatterjee and Mr. Samiran Chatterjee, who always suspected I’d end up here
ACKNOWLEDGMENTS

This dissertation is a joint endeavor; it could not have been written and successfully submitted without the help and support of great many people. I owe my gratitude to all of them who have made this dissertation possible and because of whom my graduate school experience at FSU has been one that I will cherish for many years to come.

My major professor, Dr. Piyush Kumar, didn’t know what he was getting himself into when he took me on as a student, and I will always be grateful for his support and guidance. The other members of my committee, Dr. Tyson, Dr. Mascagni and Dr. Mio, deserve hazard pay, and this thesis would not be the same without their diligence: many thanks.

My friends at FSU have motivated this work and helped me to stay grounded as I navigated through the graduate program. I will fondly cherish the friendship with my group mates, Michael, Bradley, Tathagata, Polina, and the discussions that influenced this research. I greatly value the friendship of Saikat, Chaity, Oindrila, Mayur, Himadri, Mrinal and Santosh for interesting dinner table conversations and countless number of delightful arguments ranging from philosophy to sociology to biology. I am indebted to them and to all my other friends in Tallahassee and in India for making this journey a memorable one.

Last, but not the least, I would like to thank my family for their love. Specifically, Samiparna, the love of my life, my sister-in-law, Tannistha, for their continuous and untiring moral support that made this dissertation possible. I record my deepest gratitude for my parents for always being supportive of my dreams and aspirations and for loving me for what I am. I dedicate this dissertation to them.
# TABLE OF CONTENTS

List of Tables ................................................................. viii
List of Figures ................................................................. ix
Abstract ................................................................. xiii

1 Introduction 1
  1.1 Geometric Minimum Spanning Trees ................. 2
  1.2 One Center Problem on Road Networks ................. 3
  1.3 Minimum Error Rate Training for Statistical Machine Translation 5

2 Background 7
  2.1 Preliminary Concepts for Studying the GMST Problem ........ 8
    2.1.1 Quadtrees ........................................... 8
    2.1.2 Morton Ordering .................................... 8
    2.1.3 KD Trees ........................................... 10
    2.1.4 Kruskal’s Algorithm and Union Find Data Structure ....... 12
    2.1.5 Well Separated Pair Decomposition .................... 13
    2.1.6 Bichromatic Closest Pairs ............................ 14
  2.2 Preliminary Concepts for Studying the 1-Center Problem ....... 16
    2.2.1 Single Source Shortest Path Algorithms ................. 16
    2.2.2 MultiDimensional Scaling ............................. 17
    2.2.3 Graph Partitioning .................................... 17
  2.3 Preliminary Concepts for Studying Minimum Error Rate Training . 18
    2.3.1 Classical MERT and its Shortcomings .................... 18
    2.3.2 The Translation Lattice ............................... 20

3 The GeoFilterKruskal Algorithm for Computing Geometric Minimum Spanning Trees 21
  3.1 Related Work .................................................. 23
  3.2 GeoFilterKruskal Algorithm ................................. 24
    3.2.1 Correctness .......................................... 26
    3.2.2 Analysis of the running time ........................... 26
    3.2.3 Parallelization ....................................... 31
  3.3 Experimental Setup ............................................ 32
  3.4 Experimental Results .......................................... 39
3.4.1 Comparing Quadtrees and Fair Split Trees .................................. 39
3.4.2 Synthetic Data Sets ................................................................. 46
3.4.3 Real World Data Sets ............................................................... 46
3.5 Conclusion ............................................................................... 47

4 Instant Approximate One Center On Road Networks Via Embeddings .... 48
  4.1 Related Work ........................................................................ 49
     4.1.1 The Euclidean Group Enclosing Query Problem .................. 49
     4.1.2 Network Group Enclosing Query ......................................... 51
     4.1.3 Exact Solution .................................................................. 52
  4.2 The Approximate Solution ....................................................... 53
  4.3 Measures of Distortion ............................................................ 54
  4.4 Sammon Optimization Criterion .............................................. 55
  4.5 Embedding Phase ................................................................. 55
  4.6 Handling Large Data Sets ....................................................... 61
     4.6.1 Modified algorithm for Large Data Sets ............................... 62
  4.7 Query Phase ........................................................................ 63
  4.8 Experimental Setup ............................................................. 64
  4.9 Experimental Results ............................................................ 65
     4.9.1 Interface ....................................................................... 66
  4.10 Conclusion ........................................................................ 69

5 Minimum Error Rate Training by Sampling the Translation Lattice ......... 70
  5.1 Related Work ................................................................. 70
  5.2 Sampling Candidate Translations from the Lattice ....................... 71
     5.2.1 An intuitive explanation ...................................................... 71
     5.2.2 The sampling procedure .................................................. 72
     5.2.3 Time Complexity Analysis ............................................... 73
  5.3 Experimental Results ............................................................ 75
  5.4 Analysis of results .............................................................. 75
  5.5 Conclusion ........................................................................ 79

6 Conclusions and Contributions ...................................................... 80

A National Science Foundation Innovation Corps Award ...................... 82
  A.1 Potential Commercial Impact .................................................. 82
     A.1.1 Customer Profile ........................................................... 82
     A.1.2 Customer Need .............................................................. 83
     A.1.3 How the needs are currently met? .................................... 83
     A.1.4 Payment Amount from Customers ................................... 83
  A.2 Deliverables ........................................................................ 83

References .................................................................................. 85
LIST OF TABLES

3.1 Algorithm Descriptions ........................................... 33
3.2 Point Distribution Info ............................................ 34
3.3 Surface Reconstruction Timings (Secs) .......................... 34
4.1 Time chart in seconds for computing the 1-center for query sets of sizes $2^x$, $x = 1,..,10$ ........................................... 66
5.1 Test set BLEU Scores for six different “Source-Target” Pairs .... 77
5.2 Test set BLEU Scores for the same “Source-Target” pairs using a mixed strategy combining a 100 N-best list and a random sample of size 100 after each round of decoding. ............................... 77
# LIST OF FIGURES

1.1 A Geometric Minimum Spanning Tree ........................................... 2

1.2 Example of a road network with exact ("E") and approximate ("A") 1-center computed. The shortest path from the locations to the exact center are also shown. The approximate answer is the one computed using the algorithm presented in this thesis. ........................................... 3

2.1 The Morton order curve preceding the upper left corner, and following the lower right corner of a quadtree hypercube, will never re-enter the region. ........................................... 9

2.2 The smallest quadtree hypercube containing two points will also contain all points lying between the two in Morton order. ......................... 11

2.3 A KD-tree constructed over a two dimensional point set ................... 11

2.4 Two examples of fair split .......................................................... 13

2.5 Well Separated Pair ................................................................. 13

2.6 BCCP of 9 red and 9 blue points ................................................. 14

2.7 A lattice showing some possible translations of the English sentence: *I have a blue car*. The state with ID 0 is the start state and the one with *F* is the final state. ................................................................. 20

3.1 This figure demonstrates the run time gains of the algorithm as more threads are used. We present scaling for two architectures. The AMD line was computed on the machine described in Section 4.8. The Intel line used a machine with four 2.66GHz Intel(R) Xeon(R) CPU X5355, 4096 KB of L1 cache, and 8GB total memory. For additional comparison, we include KNNG construction time using a parallel 8-nearest neighbor graph algorithm. All cases were run on 20 data sets from uniform random distribution of size $10^6$ points, final total run time is an average of the results. ................................................................. 30
3.2 Results of comparisons of tree size between fair split trees and quad trees on the same set of data. Data was one million two dimensional points taken from uniform distribution.

3.3 Results of comparisons of construction time between fair split trees and quad trees on the same set of data. Data was one million two dimensional points taken from uniform distribution.

3.4 Results of comparisons of number of well separated pairs produced as the separation factor is varied. FST is being used for the purpose. Data was one million two dimensional points taken from uniform distribution.

3.5 Results of comparisons of the error in GMST length as the separation factor is varied. FST is being used for the purpose. Data was one million two dimensional points taken from uniform distribution.

3.6 Computation Time as dimension Varies. The number of points is 500000 from uniform distribution. For each experiment 5 tests were done and the results averaged. The relative performance of the algorithms were similar for higher dimensions.

3.7 Total running time for each algorithm for varying sized data sets of uniformly random points, as the dimension increases. Data sets ranged from $10^6$ to $10^7$ points for the 2-d data. For each data set size 5 tests were done and the results averaged.

3.8 Mean run time and standard deviation to compute the GMST on data sets of various distributions (see Table 3.2) using GeoMST and GeoFK1 in 2 dimensions. The data set size was $10^6$ points. For each data set size 5 tests were done and the results averaged.

3.9 Total running time for each algorithm for varying sized data sets of uniformly random points, as the dimension increases. Data sets ranged from $10^5$ to $10^6$ points for the 3-d data. For each data set size 5 tests were done and the results averaged.

3.10 Mean run time and standard deviation to compute the GMST on data sets of various distributions (see Table 3.2) using GeoMst2 and GeoFK1 in 2 dimensions. The data set size was $10^6$ points. For each data set size 5 tests were done and the results averaged.

3.11 Total running time for each algorithm for varying sized data sets of uniformly random points, as the dimension increases. Data sets ranged from $10^5$ to $10^6$ points for the 4-d data. For each data set size 5 tests were done and the results averaged.
3.12 Mean run time and standard deviation to compute the GMST on data sets of various distributions (see Table 3.2) using Triangle and GeoFK1 in 2 dimensions. The data set size was $10^6$ points. For each data set size 5 tests were done and the results averaged. 45

3.13 Reconstructed models of a sleeping puppy, a jaguar head and DaVinci’s head. 47

4.1 Shortest path plots of query points $q_1$, $q_2$, $q_3$, $q_4$ along the edge $e$. 52

4.2 Mean distortion obtained for the Sammon and Strain criteria on Tallahassee road network data. We report the distortion only for Sammon, because the same for Strain was larger than $10^3$. The data set has 1597 nodes and 2425 edges. 56

4.3 Comparison of Sammon and Strain criteria in terms of time taken on Tallahassee road network data. We do not show the time for Strain after the 7th dimension because its execution time did not change by much beyond the 7th dimension. 57

4.4 Comparison of distortions from PC and Sammon. 58

4.5 Comparison of mean distortions from PC and Sammon. 58

4.6 Comparison of distortions from SDPT-3 and Sammon. 59

4.7 Comparison of mean distortions from SDPT-3 and Sammon. 60

4.8 Two examples of 4 point metric spaces that cannot be embedded isometrically into any Euclidean space. Each edge in both the graphs is of unit length. 60

4.9 Distortion vs K-Nearest Neighbors 61

4.10 A GEQ example. The solid points are in $Q_{de}$ and the hollow ones are in $V_{de}$. Center $c$ of $\text{MEB}(c, Q_{de})$ is snapped to its nearest neighbor (denoted by $nn(c, V_{de})$) in $V_{de}$. 64

4.11 AbsR values for 10,000 pairwise distances sampled randomly from Tallahassee, Orlando, Tampa, Florida and Miami. AbsR values are shown for 1000 pairwise distances from continental USA. 67

4.12 Error in approximation of the 1-center answer computed by our algorithm. 68

4.13 Screenshot of the Interface. The approximate one center to the query-points A, B, C, and D is shown. 69
5.1 Envelope of the set of reachable translations where the model has two feature functions \( h_1 \) and \( h_2 \). The envelope of the lattice is the outer dashed polygon, while the envelope of the N-best list is the inner one. Using the whole lattice as translation pool will result in a more marked update towards the optimal parameters. The random sample from the lattice is enclosed by the dotted line. If we use it, we can intuitively expect updates towards the optimum of intermediate effectiveness between those of the N-best list method and those of the lattice method. . . . . 72

5.2 Learning curves (BLEU on the development set) for different tested conditions for English to French (top) and French to English (bottom). . . 76
ABSTRACT

The most common approaches to analyzing algorithms are worst-case analysis, average-case analysis, and experimental analysis. In most cases, until recently, experimental analysis has not been given priority. The theoretical computer science community still continues to ignore experimental evaluation, although experimental work dominates algorithmic research in most other areas of computer science and related fields such as operations research. This thesis aims to show the importance of experimental approaches through the solution of three different problems; two in the field of computational geometry and one in the field of statistical machine translation.

The GeoFilterKruskal algorithm solves the problem of computing geometric minimum spanning trees (GMST). A common tool in tackling clustering problems, GMSTs are an extension of the minimum spanning tree graph problem, applied to the complete graph of a point set. By using well separated pair decomposition, bi-chromatic closest pair computation, and partitioning and filtering techniques, GeoFilterKruskal greatly reduces the total computation required. It is also one of the only algorithms to compute GMSTs in a manner that lends itself to parallel computation; a major advantage over its competitors.

The 1-Center problem is an important application in the field of Geographic Information Systems. Finding an exact solution to the problem is computationally expensive. In this thesis, we develop an algorithm that uses computational geometric tools to find an approximate solution to the problem, instantaneously. Exhaustive experiments on real world data sets show that our solution incurs a small constant factor approximation.

In the field of Statistical Machine Translation, the Minimum Error Rate Training (MERT) is a widely used algorithm for tuning error parameters. Most of the automated translation algorithms use the classical version of MERT, which fails to capture the underlying distribution of the translation lattice. We propose an algorithm which succeeds in overcoming this drawback, without compromising in the running time both in theory and in practice.
CHAPTER 1
INTRODUCTION

Ever since mathematics was born, there has been enormous progress in the design of algorithms, but very little of it has been put into practice and the gap between theory and practice has continuously widened over these years. Moreover, many of the recently developed algorithms are very hard to characterize theoretically and suffer from very large running-time coefficients. Thus the algorithms and data structures community needs to return to implementations as the standard of value; we call such an approach Experimental Algorithmics.

Experimental Algorithmics studies algorithms and data structures by joining experimental studies with the more traditional theoretical analyses. Experimentation with algorithms and data structures is proving indispensable in the assessment of heuristics for hard problems, in the design of test cases, in the characterization of asymptotic behavior of complex algorithms, in the comparison of competing designs for tractable problems, in the formulation of new conjectures, and in the evaluation of optimization criteria in human-related activities. Experimentation is also the key to the transfer of research results from paper to production code, providing as it does a base of well-tested implementations.

Many problems in computational geometry, particularly those related to point clouds, build solutions around nearest neighbors. This research examines two important practical applications of nearest neighbor search. Another interesting problem that this dissertation looked at is in the field of Statistical Machine Translation.

In this introductory chapter, the three distinct problems will be defined. In addition, common uses for these problems will be identified, and (hopefully) the reasons to seek more practical algorithms for solving them will be made clear. For all research subgroups, the goal will be three-fold. First, design an algorithm that can be competitive with the current state of the art. Second, complete a theoretical analysis of the algorithms runtime. The obvious goal is to improve the running time (both in theory and practice) when compared to current methods, or to match the running time with fewer theoretical restrictions on the data (such as assuming a uniform distribution). Finally, algorithms should be implemented in the most efficient manner. This will allow for a practical comparative analysis with other implementations via
experimentation, as well as allowing other interested parties to replicate experimental results.

1.1 Geometric Minimum Spanning Trees

Given a set of \( n \) points \( P \) in \( \mathbb{R}^d \), the Geometric Minimum Spanning Tree (GMST) of \( P \) is defined as the minimum weight spanning tree of the complete undirected weighted graph of \( P \), with edges weighted by the distance between their end points \( 1.1 \). In this thesis, we present a practical deterministic algorithm to solve this problem efficiently, and in a manner that easily lends itself to parallelization.

**Motivation:** GMSTs have a wide range of applications in pattern classification and clustering \([141, 53]\). One such example is the \( k \)-space clustering \([56]\) where the problem asks to divide a set of objects classified into \( k \) separate groups, the distance function being the numeric value specifying “closeness” of two objects. Spacing refers to the minimum distance between any pair of points in different clusters. The objective is to find a \( k \)-clustering such that spacing is maximized. These problems find applications in routing in mobile ad hoc networks \([134]\), identification of patterns in gene expression \([130, 90]\) and document categorization for web search \([132, 54]\).

![Figure 1.1: A Geometric Minimum Spanning Tree](image)

GMSTs can be used to compute hierarchical clusterings \([56]\). One such clustering, called the *single-linkage* clustering is essentially the well known Kruskal’s algorithm \([36]\) for computing minimum spanning trees. One forms \( v \) clusters of one object each, finds the closest pair of objects such that each object is in a different cluster, adds an edge between them and repeats this above procedure until there are exactly \( k \) clusters. This procedure is precisely Kruskal’s algorithm, stopping when \( k \) clusters are formed. Single-linkage clustering finds applications in identifying non-parametric clusters in noisy data \([135]\), astronomy \([11]\) and finding more efficient clusterings \([10]\). GMST-based clusterings are also used in deep-space surveys and simulations for the
1.2 One Center Problem on Road Networks

In this section, we introduce the 1-center problem on road networks, an important problem in Geographic Information Systems, having applications in facility location problems. The problem can be formally stated as follows: Compute a ball of minimum radius enclosing a given set of locations (latitude, longitude) on the map. The measurements on the network are done using the shortest path metric. The 1-center problem on road networks has many important applications including location-based services and facility location. Figure 1.2 shows an exact solution to the 1-center problem.

We need to distinguish between problems in which the center can be located anywhere on the road network and problems in which centers can only be located on the nodes of the network. The former category of problems are known as absolute center problems, whereas the later are known as vertex center problems. In this thesis, we implement an exact absolute 1-center as well as an exact vertex 1-center algorithm and compare each of them with an approximation algorithm that produces a vertex
center as a solution. Note that using Steiner points [41] on the edges, vertex centers can be used to approximate the absolute center solution easily.

The problem of 1-center computation has a long history and was solved exactly by Kariv and Hakimi [65] using $O(n^3)$ time for a road network with $n$ nodes (intersections). Henceforth, we will refer to this algorithm as the KH algorithm. Although algorithmic improvements in shortest path computations in the past two decades [120, 122, 123, 124] could be used to improve this running time slightly, it would still be a far cry from an instant solution. One large motivation for trying to solve the problem instantly (even though approximately), is that this could be put on the web for people to use that could compute their results interactively.

Since this is an old and well studied problem, we refer the reader to multiple books that discuss this problem [40, 94]. If $m$ is the number of query locations, it is easy to see that $\Omega(m)$ is a lower bound on this problem since without looking at the queries, it is impossible to come up with a correct solution (unless other, perhaps unrealistic assumptions are made). As far as we know, no solution for this problem that works in $O(m + \log n)$ time exists, even for approximate solutions. In this thesis, we experiment with an approximation algorithm that follows these bounds and gives reasonable approximations. The most interesting aspect of our solution is that we can cast the problem on the network to a problem in Euclidean space, a first approach to solving this problem in this fashion. This opens the door for many other similar and related problems to use these techniques and speed up approximate result computations [4, 81].

An interesting generalization of the 1-center problem is the aggregate nearest neighbor (ANN) search problem [61, 140, 103, 115, 117, 102, 81]. Given a set $P$ of interesting objects, a set $Q$ of query points, and an aggregate function $f$ (e.g., sum, max) an aggregate nearest neighbor (ANN) query retrieves the object $p$ in $P$, such that $f\{d(p, q_i), \forall q_i \in Q\}$ is minimized. The 1-center problem is a special case of the ANN problem. In our setting, the set $P$ is continuous (i.e. it is the whole road network instead of some interesting set of discrete points) and $f = \text{max}$.

ANN queries are a natural way to express requests by groups of mobile users who want to optimize their routes according to an aggregate function applying on the traveling distances. Let us consider the following real life example. Suppose a train station is being planned to be built in the city of Tallahassee. If the station is built at one of its boundaries, people living in areas in the other extreme of Tallahassee will have to drive entirely through the city to reach the station. If the traffic in the city is very heavy and congested, this will cause tremendous inconvenience for those who will have to drive such a long distance. This applies to all residents living far away from the station. In addition to the inconvenience caused to this section of the residents, this will have impacts on the transportation system of the city, in the sense that on days with a high traffic towards the station, congestions might become inevitable: flow of traffic to the other parts of the city through the roads leading to the station will also slow down. One solution to this problem is to find a location in
the city which represents the center, thus avoiding long drives for a particular section of the residents. A central location will have uniform effect on all the residents at the cost of increasing the driving distance for those living very near to the station, albeit, not by a large amount. A similar example can be selecting the location of a tourist office based on its distances to attractions in a city. ANN queries are important in geographic information systems, location-based services, navigation systems, mobile computing systems, and data mining (e.g., clustering objects in a road network) [139]. We believe that the algorithm proposed in this thesis would lead to better solutions to ANN queries in general if an instant solution is required.

Motivation: Important questions like: where should we construct a fire-station/hospital to cater to a set of people? can be answered by solving the 1-center problem on road networks. One Center queries are important in geographic information systems, location-based services, navigation systems, mobile computing systems, and data mining [103]. Another motivation to study the problem is: Given locations of people who want to meet, but want to keep the maximum distance traveled by anyone to a minimum.

1.3 Minimum Error Rate Training for Statistical Machine Translation

Minimum Error Rate Training is the algorithm for log-linear model parameter training most used in state-of-the-art Statistical Machine Translation systems\(^1\). In its original formulation, the algorithm uses N-best lists output by the decoder to grow the Translation Pool that shapes the surface on which the actual optimization is performed. Recent work has been done to extend the algorithm to use the entire translation lattice built by the decoder, instead of N-best lists.

Most state-of-the-art Statistical Machine Translation (SMT) systems are based on a log-linear model of the conditional probability of generating a certain translation given a specific source sentence. More specifically, the conditional probability of a translation \(e\) and a word alignment \(a\) given a source sentence \(f\) is modeled as:

\[
P(e, a|f) \propto \exp \left( \sum_{k=1}^{K} \lambda_k h_k(e, a, f) \right)
\]  

(1.1)

where the \(h_k(e, a, f)\) are feature functions providing complementary sources of information on the quality of the produced translation (and alignment). Once such a model is known, the decoder (i.e. the actual translation program), builds a translation by searching in the space of all possible translations the one that maximizes the conditional probability:

\(^1\)The work behind this topic was done at Xerox Research Centre Europe (XRCE) in Grenoble, France during the Fall semester of 2009, under the supervision of Dr. Nicola Cancedda. The author was a research intern at XRCE during that period.
\[(e^*, a^*) = \arg \max_{e,a} \sum_{k=1}^{K} \lambda_k h_K(e,a,f) \]  

(1.2)

where we have taken into account that the exponential is monotonic. The parameters \(\lambda_k\) determine the relative importance of the different feature functions in the global score. Best results are typically obtained by searching in the space of all possible parameter vectors \(\bar{\lambda}\) for the one that minimizes the error on a held-out development dataset for which one or more reference human translations are available, as measured by some automatic measure. This procedure is referred to as Minimum Error Rate Training (MERT).

**Motivation:** The most frequently cited benefits of statistical machine translation over traditional paradigms are:

- Better use of resources: There is a great deal of natural language in machine-readable format. Generally, SMT systems are not tailored to any specific pair of languages. Rule-based translation systems, as against SMT which is a phrase-based system [82] require the manual development of linguistic rules, which can be costly, and which often do not generalize to other languages.

- More natural translations: Rule-based translation systems are likely to result in literal translation. While it appears that SMT should avoid this problem and result in natural translations, this is negated by the fact that using statistical matching to translate rather than a dictionary/grammar rules approach can often result in text that include apparently nonsensical and obvious errors.
CHAPTER 2
BACKGROUND

In this chapter we introduce some background concepts required to study each of the problems dealt with in this thesis. We first introduce the notations, followed by the concepts.

Notations for GMST: Points are denoted by lower-case Roman letters. Dist\((p, q)\) denotes the distance between the points \(p\) and \(q\) in \(L_2\) metric. Upper-case Roman letters are reserved for sets. Scalars except for \(c, d, m\) and \(n\) are represented by lower-case Greek letters. We reserve \(i, j, k\) for indexing purposes. Vol( ) denotes the volume of an object. For a given quadtree, Box\((p, q)\) denotes the smallest quadtree box containing points \(p\) and \(q\); Fraktur letters \((a)\) denote a quadtree/fair split tree node. The Cartesian product of two sets \(X\) and \(Y\), is denoted by \(X \times Y = \{(x, y) \mid x \in X \text{ and } y \in Y\}\). We use the following notations from [18]. Let \(P\) be a point set in \(d\)-dimensional real space, denoted by \(R^d\). The bounding box of \(P\), denoted by \(R(P)\) is defined to be the smallest rectangle that encloses all points in \(P\), where the word “rectangle” denotes the Cartesian product \(R = [x_1, x'_1] \times [x_2, x'_2] \times ... \times [x_d, x'_d]\) in \(R^d\). We denote the length of \(R\) in the \(i\)th dimension by \(l_i(R) = x'_i - x_i\). We denote the maximum and minimum lengths by \(l_{\max}(R)\) and \(l_{\min}(R)\). When all \(l_i(R)\) are equal, we say that \(R\) is a \(d\)-cube, and denote its length by \(l(R) = l_{\max}(R) = l_{\min}(R)\). We write \(l_i(P), l_{\min}(P), l_{\max}(P)\) as shorthand for \(l_i(R(P)), l_{\min}(R(P))\) and \(l_{\max}(R(P))\), respectively. MinDist\((a, b)\) denotes the minimum distance between the quadtree boxes of two nodes in case of a quadtree and the same between the bounding boxes of two nodes in case of a fair split tree. Bccp\((a, b)\) computes the bichromatic closest pair of two nodes, and returns \(\{u, v, \delta\}\), where \((u, v)\) is the edge defining the Bccp and \(\delta\) is the edge length. Left\((a)\) and Right\((a)\) denotes the left and right child of a node. \(|.|\) denotes the cardinality of a set or the number of points in a quadtree/fair split tree node. \(\alpha(n)\) is used to denote inverse of the Ackermann function [36].

Notations for 1-Center: A network is an undirected graph \(G = (V, E, w)\) where \(V\) is the set of vertices (or nodes) and \(E\) is the set of edges, and \(w : E \rightarrow \mathbb{R}^+\) associates each edge to a positive real number. This number is the weight or cost of the edge. If \(V = \{v_1, v_2, ..., v_n\}\) and there is an edge \(e \in E\) such that \(v_1\) and \(v_2\) are its two endpoints, then we represent \(e\) as \((v_1, v_2)\). Let \(Q\) be a set of query points
\(\{q_1, q_2, ..., q_m\}\) (e.g., users) on the network\(^1\). In this thesis, using the longitude and latitude of each vertex, the Euclidean length of each edge is computed. We use this distance as the weight of each edge. The 1-center problem asks for a point \(p\) in the graph, such that the maximum distance of \(p\) to all points in \(Q\) is minimized, when traveling along the edges. We denote the shortest path distance between two vertices \(v_i\) and \(v_j\) by \(d_{ij}\), \(\forall i, j = 1, 2, ..., n\). In this thesis, we assume that the road network is undirected. This assumption enables us to embed the network in a Euclidean space. We denote the Euclidean space by \(\mathbb{R}^d\), where \(d\) is the dimension of the space and the embedded set of vertices by \(V'\). Let \(x_i \in V'\) denote the vertex corresponding to \(v_i\). The Euclidean distance between \(x_i\) and \(x_j\) is denoted by \(d'_{ij} = ||x_i - x_j||\), i.e. the Euclidean 2-norm or the \(L_2\) norm.

**Notations for MERT:** Small Greek letters are used to denote scalars. Subscripts of \(n\) denote vertices of the translation lattice. Subscripts of \(e\) denote edges of a translation lattice. Some more notations will be introduced as the discussion on the topic progresses.

## 2.1 Preliminary Concepts for Studying the GMST Problem

Below we enumerate the required tools.

### 2.1.1 Quadtrees

One of the first types of spatial subdivision trees, the quadtree, and an algorithm to construct it was first introduced in 1975 by J. Bentley [48]. A quadtree is defined by partitioning space into \(2^d\) equal, axis-aligned hypercubes. Each subspace is then further partitioned until all points in the data set are contained within their own-hypercube. Naively, it can be constructed in \(O(n \log n)\) time by first sorting all input points according to their Morton order (see section 2.1.2), then computing the smallest quad-tree boundary lying between any two adjacent points. This constructs a tree containing \(O(n)\) elements. However, it has since been shown that an equivalent tree can be constructed in \(O(n)\) time, by reducing the problem to a Cartesian tree [25]. Nearest neighbor algorithms based on quad-trees have been theorized, with expected running times similar to those based on other trees, such as kd-trees (see 2.1.3). In general, they are thought to be less efficient in practice [13, 23].

### 2.1.2 Morton Ordering

Morton order is a space filling curve. It reduces points in \(\mathbb{R}^d\) to a one dimensional space, while attempting to preserve locality. The Morton order curve can be

---

\(^1\)We divide each edge into enough number of vertices such that the queries can be virtually anywhere on an edge or on a vertex of the network.
conceptually achieved by building a quadtree, then recursively ordering the hypercubes. Morton order curves are sometimes called Z curves because these hypercubes are ordered so as to form a Z.

In practice, Morton order can be determined by computing the Z-value for all the data points, then ordering them. The Z-value is a bitwise operation on the coordinates of a point. For a particular point, the bits of the coordinate values are interleaved. The resulting number is the Z-value. By constructing the curve in this way, we can implicitly define a quadtree.

Chan [24] showed that the relative Morton order of two integer points can be easily calculated, by determining which pair of coordinates have the first differing bit in binary notation in the largest place. He further showed that this can be accomplished using a few binary operations. Using this method, Morton order can be reduced to sorting. A method for computing relative Morton order on floating points is proposed by Connor and Kumar [33].

Morton order curves have two simple properties that are useful in nearest neighbor searching. The first, shown in figure 2.1, is that the curve does not double back on itself. Once the Morton order leaves a hypercube of the quadtree, it will not intersect that hypercube again. The second property, shown in figure 2.2, states that a quadtree hypercube containing two points on the Morton order curve will also contain all points which lie on the curve between the original two.

**Compressed Quadtrees:** Let \( p_1, p_2, ..., p_n \) be a given set of points in Morton order. Let the index \( j \) be such that \( \text{Vol}(\text{Box}(p_{j-1}, p_j)) \) is the largest. Then the compressed quadtree \( Q \) for the point set consists of a root holding the \( \text{Box}(p_{j-1}, p_j) \) and two subtrees recursively built for \( p_1, ..., p_{j-1} \) and \( p_j, ..., p_n \). Note that this tree is simply the
Cartesian tree [51] of $\text{Vol}(\text{Box}\{p_1, p_2\}), \text{Vol}(\text{Box}\{p_2, p_3\}), \ldots, \text{Vol}(\text{Box}\{p_{n-1}, p_n\})$. A Cartesian tree is a binary tree constructed from a sequence of values which maintains three properties. First, there is one node in the tree for each value in the sequence. Second, a symmetric, in-order traversal of the tree returns the original sequence. Finally, the tree maintains a heap property, in which a parent node has a larger value than its child nodes. The Cartesian tree can be computed using a standard incremental algorithm in linear time, given the ordering $p_1, p_2, \ldots, p_n$ [51, 25]. Hence, for both integer as well as floating point coordinates, the compressed quadtree $Q$ can be computed in $O(n)$ time excluding the time for sorting the points in Morton order, which takes $O(n \log n)$ [33] time for floating point coordinates. In our experimental results, we compare the effectiveness of using a compressed quadtree to results using a fair split tree. Empirically, we show that while the compressed quadtree can offer the same theoretical guarantees, the fair split tree is superior in practice. We will show later that the compressed quadtree does allow for theoretically linear time construction of GMSTs.

2.1.3 KD Trees

As an improvement over the quad-tree, Bentley introduced the kd-tree [13]. This is another spatial decomposition tree, which relaxes the requirement of the quadtree that all regions be divided equally. Instead, areas are divided into axis-aligned hyperrectangles. The method for choosing the splitting point varies, but the goal is always to divide the points as evenly as possible, while simultaneously attempting to keep the ratio of the side lengths as low as possible. A common method for choosing the split is to rotate through the dimensions in the point set. That is, the first level of the tree is formed by sorting points according to their x coordinate, then cutting it with an orthogonal plane at the median point. The next level would be split similarly on the y axis, and so on.

It is worth mentioning that the run-time of nearest neighbor algorithms based on these trees is related to both the height of the tree, and the aspect ratio of the hyperrectangles. If the rectangles are too skinny, nearest neighbor balls will intersect more than a constant number, thus inflating the running time.

One improved method for construction of kd-trees modifies the way the splitting plane is chosen. Instead of merely rotating through the dimensions, the splitting dimension at each internal node is determined by the spread of points in that dimension. This, in practice, helps to bound the aspect ratio of the hyperrectangles. This is accepted as the standard kd-tree splitting rule, and guarantees a tree of height $O(\log n)$ [50]. It does not, however, ensure that the hyperrectangles are well-formed (having a constant factor aspect ratio).

Advanced Splitting Rules. Many improvements have been made to the implementation of KD-trees, based on the way the hyperrectangles are split. Some of
Figure 2.2: The smallest quadtree hypercube containing two points will also contain all points lying between the two in Morton order.

Figure 2.3: A KD-tree constructed over a two dimensional point set

these include the Midpoint Rule [50], the Sliding Midpoint Rule [86], the Balance Split Rule [7] and the Hybrid Split Rule [7].

The Midpoint Rule dictates that rather than splitting the hyperrectangles along the dimension of greatest spread, they should be split by a plane orthogonal to the midpoint of the longest side. This begins to address the need to keep the aspect ratio a constant, but can lead to a tree of arbitrary height, as well as empty internal nodes.

The Sliding Midpoint Rule is a practical modification of the Midpoint Rule, introduced by Arya and Mount in their ANN nearest neighbor library [6]. In the case of an empty hyperrectangle, the split is shifted to lie on the nearest point contained in the hyperrectangle. This eliminates the risk of empty nodes. It can be shown that while the height of this tree could still be as bad as $O(n)$, nearest neighbor searches done on it work very well in practice, and have an expected running time of $O(\log n)$ [7].

The Balance Split Rule is a further relaxation of the Sliding Midpoint Rule. Again,
the longest side of the hyperrectangle is chosen to be split. This time, however, an orthogonal splitting plane is chosen so that the number of points on either side are roughly balanced. This yields an expected query time of $O(\log n)$, and has a worst case query time of $O(\log^d n)$.

The Hybrid Split Rule alternates between the Sliding Midpoint Rule and the Balance Split Rule at different levels of the tree in an attempt to limit both the aspect ratio, and the height of the tree. This rule yields a tree with a height of $O(\log n)$, while still having rectangles of bounded aspect ratio, which gives an expected running time of $O(\log^d n)$.

**Fair Split Tree.** Given a point set $P$, define a split of $P$ as two non-empty point sets lying on either side of an axis-aligned hyperplane, called the splitting hyperplane, perpendicular to one of the coordinate axes and not containing any point in $P$. The split tree of $P$ is defined to be a binary tree and is constructed recursively as follows. If $|P| = 1$, its unique split tree consists of single node $P$. Otherwise, the split tree is any tree with the root node $P$ and two subtrees that are split trees of the subsets formed by a split of $P$. For any node $a$, we denote its parent by $p(a)$, except the root node which does not have a parent. The outer rectangle of $a$, denoted by $\hat{R}(a)$, for each node $a$ is defined top down as follows:

- For the root node $P$, let $\hat{R}(P)$ be an open $d$-cube centered at the center of $R(P)$, with \( l(\hat{R}(P)) = l_{\text{max}}(P) \).

- For all other $a$, the splitting hyperplane used for the split of $p(a)$ divides $\hat{R}(p(a))$ into two open rectangles. Let $\hat{R}(a)$ be the one that contains $a$.

A *fair split* of $a$ is defined to be a split of $a$ in which the splitting hyperplane is at a distance of at least \( l_{\text{max}}(a)/3 \) from each of the two boundaries of $\hat{R}(a)$ parallel to it. See [18] for details. A split tree formed using only fair splits is called a *fair split tree* (FST).

One important factor here is the following: it might happen that each split can result in all but one point going to one side, so a naive implementation could have a quadratic complexity. In our implementation, we used a method suggested by Callahan [18]. He presents an algorithm to construct a fair split tree in $O(n \log n)$ time. Instead of computing a single fair split at each call, he constructs what is called a *partial fair split tree* and then computes the final FST.

### 2.1.4 Kruskal’s Algorithm and Union Find Data Structure

Published in 1956, Kruskal’s algorithm [71] finds the minimum weight spanning tree of a graph. It begins by placing the edges in the graph into a priority queue, ordered by weight. Then it builds a structure (commonly called Union Find) which will identify edges in terms of connected components. Edges are removed from the head of the priority queue and inserted into the MST if they do not create a cycle.
This proceeds until \( n - 1 \) edges have been added (where \( n \) is the number of vertices in the graph).

While this algorithm was designed to work for arbitrary graphs, it can be used to find the MST built on a set of points. In this case, the original graph is considered to be the complete graph of the point sets, and the weight to be the Euclidean distance between them. However, since there are \( O(n^2) \) edges in the complete graph, this is an expensive approach. Sections 2.1.5 and 2.1.6 describe tools used to improve the viability of this method, and chapter 3 details a new algorithm for efficiently using Kruskal's method to find geometric minimum spanning trees.

The \texttt{UnionFind} data structure \cite{36} maintains a set of partitions indicating the connectivity of points based on the edges already inserted into the GMST. Given a \texttt{UnionFind} data structure \( G \), and \( u, v \in P \subseteq \mathbb{R}^d \), \( G \) supports the following two operations: \( G.\text{Union}(u, v) \) updates the structure to indicate the partitions containing \( u \) and \( v \) belong to the same connected component; \( G.\text{Find}(u) \) returns the node number of the representative of the partition containing \( u \). Our implementation also does union by rank and path compression. A sequence of \( m \) \( G.\text{Union}() \) and \( G.\text{Find}() \) operations on \( n \) elements takes \( O(m\alpha(n)) \) time in the worst case. For all practical purposes, \( \alpha(n) \leq 4 \) (see \cite{36}).

### 2.1.5 Well Separated Pair Decomposition

![Figure 2.5: Well Separated Pair](image)

Proposed by Callahan \cite{18}, the well-separated pair decomposition uses a spatial decomposition tree to create a simplified representation of the complete graph of a
point set. In essence, the \( n^2 \) edges of the complete graph are represented by \( O(n) \) components. Formally, given a spatial decomposition tree \( Q \), built on a set of points \( P \) in \( \mathbb{R}^d \), the well separated pair decomposition (WSPD) of the tree consists of all pairs of nodes \( \{(a_1, b_1), ..., (a_m, b_m)\} \) such that for every point \( p \in a_i \) and every point \( q \in b_i \), \( \text{MinDist}(a_i, b_i) < \gamma \ast \text{dist}(p, q) \). \( \gamma \) is known as the separation factor.

The main properties of WSPD are:

- Every pair of points \((p, q)\) should appear in exactly one well separated pair (WSP).
- For each pair \((a_i, b_i)\), there are no points common to both \( a_i \) and \( b_i \).

Construction of the WSPD can be executed on a quadtree or fair split tree in \( O(n) \) time, and Callahan proved that construction will yield \( O(n) \) WSPs. Note that while the process to construct the WSPD is identical regardless of the tree used, the number of WSPs can vary by a substantial amount (although within a constant factor). For example, for uniform distributions in 2 dimensions, on average, the quadtree based approach produces 27% more WSPs. Note that the process of constructing the WSPD is identical regardless of whether or not a compressed quadtree is used, or a fair split tree (or, in fact, any spatial decomposition tree). Section 2.1.6 describes Bichromatic closest pair computation, which can be used along with WSPD to compute geometric minimum spanning trees. A new algorithm to compute the GMST using these methods is presented in chapter 3.

### 2.1.6 Bichromatic Closest Pairs

![BCCP of 9 red and 9 blue points](image)

Figure 2.6: BCCP of 9 red and 9 blue points

Given two sets of points, \( A \) (colored red) and \( B \) colored green, the bi-chromatic closest pair (BCCP) of \((A, B)\) is defined as the minimum weight edge with endpoints \( p \in A \) and \( q \in B \). Callahan [18] showed that given a WSPD, the geometric minimum spanning tree is a subset of its BCCP edges (Lemma 2.1.1).

**Lemma 2.1.1** The set of all the BCCP edges of the WSPD contain the edges of the GMST.

In fact, GMST computation can be reduced to the computation of the BCCPs for a WSPD. While faster BCCP algorithms exist [3], in practice a simple quadratic
algorithm for finding BCCPs is typically used in GMST construction. This is due to the fact that competitive algorithms for GMST seek to minimize the number of BCCP computations necessary, and, in most cases, rarely have to compute BCCPs on WSPs of greater than a constant size. A simple, recursive BCCP algorithm is presented as in Algorithm 1.

Given a WSP, Q, and nodes from the tree, a and b, we compute the distance between the nodes, and recurse if that distance is less our current minimum. Clearly, MinDist(a, b) is a lower bound on the bichromatic closest pair distance of A and B (where A and B are the point sets contained in a and b respectively). This is repeated until we have a minimum distance between one pair of points, p ∈ a and q ∈ b.

<table>
<thead>
<tr>
<th>Algorithm 1 BCCP Algorithm [98]: Compute {p', q', δ'} = Bccp(a, b, δ)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Require:</strong> If δ is not specified, η is an upper bound on δ.</td>
</tr>
<tr>
<td><strong>procedure</strong> Bccp(a, b, δ)</td>
</tr>
<tr>
<td>if (</td>
</tr>
<tr>
<td>Let p' ∈ A, q' ∈ B</td>
</tr>
<tr>
<td>if Dist(p', q') &lt; δ then return {p', q', Dist(p', q')}</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>if Vol(a) &lt; Vol(b) then Swap(a, b)</td>
</tr>
<tr>
<td>γ = MinDist(Left(a), b)</td>
</tr>
<tr>
<td>ζ = MinDist(Right(a), b)</td>
</tr>
<tr>
<td>if γ &lt; δ then</td>
</tr>
<tr>
<td>{p, q, δ} = Bccp(Left(a), b, δ)</td>
</tr>
<tr>
<td>if ζ &lt; δ then</td>
</tr>
<tr>
<td>{p, q, δ} = Bccp(Right(a), b, δ)</td>
</tr>
<tr>
<td>return {p, q, δ}</td>
</tr>
<tr>
<td><strong>end procedure</strong></td>
</tr>
</tbody>
</table>

The input to algorithm 1 are two nodes a, b ∈ Q, that contain sets A, B ⊆ P and a positive real number δ, which is used by the recursive calls in the algorithm to keep track of the last closest pair distance found. The output of the algorithm is the closest pair of points (p, q), such that p ∈ A and q ∈ B with minimum distance δ = Dist(p, q). Initially, the BCCP distance is equal to η, where, in case of a quadtree, η represents the distance between the last point in the Morton order of A and the first point in the Morton order of B, assuming without loss of generality all points in A are smaller than all points in B in Morton order. In case of a FST, η is set to ∞. If both A and B are singleton sets, then the distance between the two points is trivially the Bccp distance. Otherwise, we compare Vol(a) and Vol(b) and compute the distance between the lighter node and each of the children of the heavier node. If either of these distances is less than the closest pair distance computed so far, we recurse on the corresponding pair. If both of the distances are less, we recurse on
both of the pairs.

### 2.2 Preliminary Concepts for Studying the 1-Center Problem

Below we discuss the necessary tools.

#### 2.2.1 Single Source Shortest Path Algorithms

In the single-source shortest path problem, we have to find shortest paths from a source vertex \( v \) to all other vertices in the graph. The most popular and widely used algorithm for this problem is the well known Dijkstra’s algorithm [36]. Given a source \( q_s \) and a destination node \( q_d \), the algorithm expands the network from \( q_s \) until \( q_d \) is reached. Starting the traverse from the source node, the algorithm maintains a priority queue \( Q \) to organize the neighbors of the nodes found so far, such that the intermediate nodes from \( q_s \) to \( q_d \) are visited according to their increasing distances from \( q_d \). The main drawback of this algorithm is that it may visit many nodes far from the shortest path. Different modifications to the Dijkstra’s algorithm are available in literature. In the next section we introduce some important shortest path query algorithms.

**Shortest Path Query Algorithms.** The above mentioned shortcoming of Dijkstra’s algorithm can be alleviated by using lower bounds on distances. The \( A^* \) search algorithm [116] uses this trick by using Euclidean distance as the lower bound. Let \( d_e(q_i, q_j) \) be the Euclidean distance between \( q_i \) and \( q_j \) and it lower bounds the network distance \( d(q_i, q_j) \). The \( A^* \) search organizes the nodes \( q_i \) to be visited by

\[
F_d(q_i) = d(q_s, q_i) + d_e(q_i, q_d).
\]

\( F_d(q_i) \) restricts the shortest path distance from \( q_s \) to \( q_d \) via \( q_i \). The node with the minimum \( F_d(q_i) \) is visited next and its neighbors are added to the priority queue \( Q \). The process terminates when the destination node \( q_d \) is dequeued from \( Q \). Shortest path search can be accelerated by materializing the network distance between every pair of nodes [140]. The high storage cost of fully materialized distances makes this approach infeasible even for networks of moderate sizes. \( H_i T_i \) [62] and \( HEPV \) [63] alleviates the extreme space requirements by partial materialization. \( H_i T_i \) first partitions the network into subgraphs that are small enough to fit into the memory. These subgraphs can be abstracted as network nodes which are recursively grouped at the higher level. At each level, all the edges that connect boundary nodes of different subgraphs at the lower level are explicitly stored together with the corresponding distance. To compute the shortest path between two given nodes, it suffices to find the most detailed subgraphs that contain the nodes and use materialized information stored in the higher level nodes of the two search paths. \( HEPV \) performs a similar hierarchical partitioning but precomputes and stores more network distances.
Efficient practical methods to process shortest path queries are often devised by following a feedback loop that consists of design, analysis, implementation and experimentation. For a detailed review of these algorithms and approximate shortest path queries, please refer to [120].

2.2.2 MultiDimensional Scaling

Multidimensional Scaling (MDS) is a family of methods for embedding metric spaces into low dimensional Euclidean spaces. For a detailed review of different embedding techniques, the reader is referred to a survey [84] and Hanan Samet’s book [112]. In this thesis, we compare our choice of MDS criterion with the classical method and the Place Center algorithm [2], the most recently available MDS method in literature. The aim of this work in the thesis is not to devise a new embedding algorithm. Rather, we show that real world road networks can be embedded into a low dimensional Euclidean space (to a reasonable approximation) and how this embedding can be used to solve the problem of our interest by using computational geometric tools.

Bounding Errors for Embedding of Graphs: The problem of embedding road network graphs seeks a method that fixes a distortion bound for every pair of distances and asks for the minimum dimension a data set can be embedded in while maintaining this distortion. A celebrated theorem of [16] states that every graph $X$ with a finite number of vertices embeds into Euclidean space with distortion $O(\log n)$. A fundamental question of graph embedding is whether its metric dimension is related to its intrinsic dimension. A good measure of intrinsic dimension is the doubling constant of the graph. The doubling constant $\lambda$ of a graph is the minimum $\lambda$ such that every ball with the center at any vertex of the graph can be covered by $\lambda$ balls of half the radius [1]. It can be shown that any $n$ vertex graph $X$ embeds into Euclidean space with dimension $O(dim(X)/\epsilon)$ and distortion $O(\log^{\frac{1}{1+\epsilon}} n)$, where $dim(X)$ is the doubling dimension of $X$ [1]. For a detailed literature review, see [1, 60]. All of these algorithms only provide an upper bound and thus do not rule out chances of good embeddings. One of the drawbacks of MDS is that it becomes too computational and memory intensive to embed graphs of large sizes using implementations currently available in literature. Thus, one has to look into other ways of embedding large graphs, one of them is the divide and conquer approach. We will explain our method in the next section and use it in chapter 4. Before that, we introduce the concept of graph partitioning.

2.2.3 Graph Partitioning

We were required to partition the road network graph when we ran our experiments on data sets of size too large to be handled by MDS. We partitioned the graph into smaller pieces each of size small enough such that they can be embedded by MDS.
In mathematics, the graph partitioning problem consists of dividing a graph into pieces, such that the pieces are of about the same size and there are few connections between the pieces. Given a graph $G$ and an integer $k > 1$, partition $V$ into $k$ parts (subsets) $\{V_1, V_2, ..., V_k\}$ such that the parts are disjoint and have equal size, and the number of edges with endpoints in different parts is minimized. In practical applications, a small imbalance $\epsilon$ in the part sizes is usually allowed, and the balance criterion is

$$\max |V_i| \leq (1 + \epsilon) \frac{|V|}{k}$$  \hspace{1cm} (2.1)

Graph partitioning is known to be NP-complete, but can be solved in polynomial time for $|V_i| = 2$ by matching [52]. Several practical methods for graph partitioning are available in the literature [66]. In this thesis, we developed our own partitioning scheme based on our requirements. We embed the network in a high dimensional Euclidean space and use the Euclidean distance in that space as an estimate of the shortest path distance in the network. This estimate tends to be worse for nodes which are far apart. Suppose we have two nodes with a large shortest path distance in the same partition. MDS embeds these two points in the Euclidean space with some distortion. When we use MDS to estimate the distance between two points in two different partitions, there is an error involved in the estimation of the embedding distance due to the least squares method. The error in estimate of the shortest path distance between two points in two different partitions is a combined effect of the distortion obtained from MDS and the error involved in FastMDS (see chapter 4). To minimize this problem, we partitioned the graph in such a way that the pairwise shortest path distances between nodes in each partition is less than the same between two nodes in different partitions. In this way, the distortion of MDS is less, resulting in a smaller distortion when FastMDS comes into effect. We discuss our scheme in more detail when we explain our algorithm.

### 2.3 Preliminary Concepts for Studying Minimum Error Rate Training

Below we describe the background concepts.

#### 2.3.1 Classical MERT and its Shortcomings

The most widespread MERT algorithm is the one developed by Och [99]. This algorithm starts by initializing the parameter vector $\lambda$. For each source sentence in the development set, the decoder is used to initialize a translation pool with a list of N-best scoring candidate translations according to the model. Using this pool and the corresponding reference translations then, an optimization procedure is run to update the parameter vector to a $\lambda'$ with reduced error. The decoder is then invoked again, the new output N-best list is merged into the translation pool, and the procedure is
iterated. The algorithm stops either after a predefined number of iterations or upon convergence, which is reached when no new element is added to the translation pool of any sentence, or when the size of the update in the parameter vector is below a threshold.

The error measure minimized at each iteration is usually BLEU [104]. BLEU essentially measures the precision with which the translation produced by a system recovers n-grams [82] of different orders from the available reference translation(s), used as a gold standard.

The optimization procedure that is run within each iteration on the growing translation pools is based on the key observation that BLEU only depends on the single translation receiving the highest score by the translation model (which would be the one shown to the recipient) in the translation pool. This in turn means that, for any given sentence, its contribution to BLEU changes only when the value of the parameters change in such a way that the sentence ranking first according to the model switches from one to another. This situation does not change when one considers all the sentences in a development set instead of just one: while varying the $\lambda$ vector, the BLEU score changes only when there is a change at the top of the ranking of the alternatives for at least one sentence in the set. In other words, BLEU is piecewise constant in $\lambda$. MERT then proceeds by performing an iterative line search by fixing each time the value of all components of $\lambda$ but one$^2$: for such a free parameter a global optimum can be identified by enumerating all the points that cause a change in BLEU. The value of the component is then fixed at the middle of an interval with maximum BLEU, and the procedure is iterated until convergence. Since the error function is highly irregular, and the iterative line search is not guaranteed to converge to a global optimum, the procedure is repeated many times with different initializations, and the best convergence point is retained.

The MERT algorithm suffers from the following problem: it assumes at each iteration that the set of candidates with a chance to make it to the top (for some value of the parameter vector) is well represented in the translation pool. If the translation pool is formed in the standard way by merging N-best lists, this assumption is easily violated in practice. Indeed, the N-best list often contains only candidates displaying minor differences, and represents only a very small sample of alternative possible translations, strongly biased by the current parameter setting.

Recognizing this shortcoming, Macherey et al. [85] extended the MERT algorithm so as to use the whole set of candidate translations compactly represented in the search lattice produced by the decoder, instead of only a N-best list of candidates extracted from it. This is achieved via an elegant but relatively heavy dynamic programming algorithm that propagates sufficient statistics (called envelopes) throughout the whole search graph. The reported theoretical worst-case complexity of this algorithm is

$^2$More generally, one can select each time a combination of coordinates identifying a line in the parameter space, and is not restricted to a coordinate direction.
$O(|V||E| \log |E|)$, where $V$ and $E$ are the vertex set and the edge set of the lattice respectively.

### 2.3.2 The Translation Lattice

Finding the optimal translation according to Equation 1.1 is NP-complete [68]. Most phrase-based SMT systems resort then to beam-search heuristic algorithms for solving the problem approximately. In their most widespread version, PBSMT decoders [82] proceed by progressively extending translation prefixes by adding one new phrase at a time, and correspondingly “consuming” portions of the source sentence. Each prefix is associated with a node in a graph, and receives a score according to the model. Whenever two prefixes having exactly the same possible extensions are detected, the lower-scoring one is merged into the other, thus creating a re-entrancy in the directed graph, which has then the characteristics of a lattice (Figure 2.7). Edges in the lattice are labelled with the phrase-pair that was used to perform the corresponding extension, the source word positions that were covered in doing the extension, and the corresponding increment in model score.

![Figure 2.7: A lattice showing some possible translations of the English sentence: *I have a blue car.* The state with ID 0 is the start state and the one with $F$ is the final state.](image)

In chapter 5 we propose a MERT method consisting in growing the translation pool using samples randomly drawn from the translation lattice. We empirically measure a systematic improvement in the BLEU scores compared to training using N-best lists, without suffering the increase in computational complexity associated with operating with the whole lattice.
CHAPTER 3

THE GEOFILTERKRUSKAL
ALGORITHM FOR COMPUTING
GEOMETRIC MINIMUM SPANNING
TREES

In this chapter, a practical deterministic algorithm to solve the problem of constructing geometric minimum spanning trees is presented. Called GeoFilterKruskal [27], the algorithm efficiently computes the minimum spanning tree of a point set $P$ using well separated pair decomposition in combination with a simple modification of Kruskal’s algorithm. When $P$ is sampled from uniform random distribution, we show that our algorithm takes one parallel sort plus a linear number of additional steps, with high probability, to compute the minimum spanning tree. Experiments show that our algorithm works better in practice for most data distributions compared to the current state of the art [98]. Our algorithm is easy to parallelize and to our knowledge, is currently the best practical algorithm on multi-core machines for dimensions greater than 2.

We use a compressed quad-tree/fair split tree to build a well separated pair decomposition (WSPD) [18], followed by a sorting based algorithm similar to Filter-Kruskal [100]. This opens up the possibility of filtering out well separated pairs with a large number of points, before it becomes necessary to calculate their bichromatic closest pair. Additionally, we can compute the bichromatic closest pair of well separated pairs of similar size in batches.

We analyze the theoretical running time of our algorithm when points are sampled from a uniform distribution. In this chapter, when we talk about the theoretical running time of our algorithm, the underlying distribution should be assumed as uniform unless otherwise stated.

The fact that our algorithm takes one parallel sort plus a linear number of additional steps, with high probability, to compute the GMST, is an improvement over the original Filter-Kruskal algorithm [100]. The expected running time for constructing the MST for arbitrary graphs with random edge weights, using the original Filter-
Kruskal algorithm \cite{100} is $O(m + n \log n \log m/n)$, where $m$ and $n$ are the number of edges and vertices of the graph respectively.

If the coordinates of the points in the input set are of size $O(\log n)$ ($n$ being the size of the point set), and the word size of the machine is greater than or equal to $\log n$, the running time of our algorithm is $O(n)$ if we use radix sort \cite{49, 25}. Linear time GMST algorithms exist in literature \cite{31, 14, 44, 109}. All these approaches use bucketing techniques to execute a spiral search procedure for finding $k$-nearest neighbors of each point in order to build a supergraph of the GMST with $O(n)$ edges. From the supergraph, the GMST is computed. Unfortunately, in our experiments, finding $k$-nearest neighbors for every point, even when $k = O(1)$, proved to be as expensive as finding the actual GMST. Under the above mentioned assumptions, for uniform distributions, we achieve the same runtime complexity as the linear time algorithms, but without suffering from the drawback of the bucketing technique that makes them impractical to implement for other distributions.

The algorithm proposed in this thesis is the first one available in literature that allows the scope for parallelization. The construction of the WSPD can be parallelized. The ability to calculate bichromatic closest pair in batches allows parallel execution of large portions of the algorithm. Assuming a CRCW PRAM model with $O(\log n)$ processors, if sorting is allowed to be done in parallel (mergesort), the running time is $O(n)$, regardless of the assumptions on the word-size of the machine and the coordinates of the input points.

Narasimhan et. al \cite{98} proposed a practical algorithm (GeoMST2) which is the best available method for computing GMSTs till date. Thus all our results, both experimental and theoretical, were compared against GeoMST2. We summarize the contributions of the thesis for the problem below.

1. Our algorithm shows significant runtime improvements over GeoMST2 for two and higher dimensions.

2. The algorithm is easy to parallelize unlike GeoMST2.

3. In contrast with GeoMST2 which is inherently $O(n \log n)$, the running time of our quadtree based algorithm is upper bounded only by one sorting time which improves to $O(n)$ under the assumptions mentioned above. The introduction of the new data structure upper bounds the preprocessing stage of our algorithm by $O(n \log n)$, although the MST construction part is still linear if the coordinates of the points obey the aforementioned assumption.

4. Our algorithm is faster compared to Filter-Kruskal on geometric instances.

5. We have provided a parallel implementation of the well separated pair decomposition on a compressed quadtree, which can be computed in $O(n)$ time \cite{25, 51}.

The code is a part of the STANN library \cite{34} and is available for download. At the time of writing of this thesis we are not aware of any other open source implementation of
the well separated pair decomposition using a compressed quadtree. For comparison purposes, we worked with the distributions on which GeoMST2 was run [98].

The rest of the chapter is organized as follows. Section 3.1 discusses prior art on the problem, section 3.2 provides details of our algorithm and its analysis. Finally sections 3.3 and 3.4 provides the experimental results followed by conclusion in section 3.5. The remaining sections provide a detailed description of the algorithm and experiments.

3.1 Related Work

It is well established that the GMST is a subset of edges in the Delaunay triangulation of a point set [106]. It is well known that that this method is inefficient for any dimension $d > 2$. It was shown by Agarwal et al. [3] that the GMST problem is related to solving bichromatic closest pairs for some subsets of the input set. The bichromatic closest pair (BCCP) problem is defined as follows: given two sets of points, one red and one green, find the red-green pair with minimum distance between them [72]. Callahan [18] used well separated pair decomposition and bichromatic closest pairs to compute the MST in $O(T_d(n, n) \log n)$, where $T_d(m, n)$ is the time required to solve the bichromatic closest pairs problem for $m$ red and $n$ green points in $d$ dimensions.

The relative hardness of the BCCP and GMST problems have been studied by Erickson [45]. Let $\Pi_1$ and $\Pi_2$ be two problems with run time complexities $T_1(n)$ and $T_2(n)$, where $n$ is the size of the input. If $T_2(n) = O(T_1(n))$, then we say that $\Pi_1$ is harder than $\Pi_2$. If $T_2(n) = O(T_1(n) \log^c(n))$ for some constant $c$, then we say that $\Pi_1$ is almost harder than $\Pi_2$. Typically, this means that $\Pi_2$ is solved by a binary search or parametric search [32, 91] using an algorithm for $\Pi_1$ as an oracle, although more complicated reductions are also possible. Finally, we say that $\Pi_1$ is probably harder than $\Pi_2$ if $\Pi_1$ is almost harder than $\Pi_2$, and $T_1(n) = \Omega(n^{1+\delta})$ for some $\delta > 0$ implies that $\Pi_1$ is harder than $\Pi_2$. Erickson showed that the GMST problem is harder than bichromatic closest pair problem, and bichromatic closest pair is probably harder than computing the GMST [45].

Clarkson [31] gave an algorithm that is particularly efficient for computing GMST for points that are independently and uniformly distributed in a unit $d$-cube. His algorithm has an expected running time of $O(n\alpha(cn, n))$, where $c$ is a constant depending on the dimension and $\alpha$ is an extremely slow growing inverse Ackermann function [36]. Bentley [14] also gave an expected nearly linear time algorithm for computing GMSTs in $\mathbb{R}^d$. Dwyer [44] proved that if a set of points is generated uniformly at random from the unit ball in $\mathbb{R}^d$, its Delaunay triangulation has linear expected complexity and can be computed in expected linear time. Since GMSTs are subsets of Euclidean Delaunay triangulations, one can combine this result with linear time MST algorithms [67] to get an expected $O(n)$ time algorithm for GMSTs of uniformly distributed points in a unit ball. Rajasekaran [109] proposed a simple expected linear time algorithm to compute GMSTs for uniform distributions in $\mathbb{R}^d$. 

23
All these algorithms use the spiral search technique which does not work efficiently for distributions other than uniform. We discuss this in Section 4, and show some experimental results in Section 6.

Narasimhan et al. [98] gave a practical algorithm that solves the GMST problem. They prove that for uniformly distributed points, in fixed dimensions, an expected $O(n \log n)$ steps suffice to compute the GMST using well separated pair decomposition. Their algorithm, GeoMST2, mimics Kruskal’s algorithm [71] on well separated pairs and eliminates the need to compute bichromatic closest pairs for many well separated pairs. At the time of submission of this work for publication [27], to our knowledge, this implementation is the state of the art, for practically computing GMSTs in low dimensions ($d > 2$). However, improvements to GeoMST2 [98] have been announced since then [88]. While we were aware of a preliminary version of this paper [87], exhaustive experimental results were lacking in this short announcement. In the detailed version of the paper the authors used a dual tree based Boruvka’s algorithm (DTB) [36] to compute the minimum spanning tree. They also show an $O(n \log n)$ running time construction of a GMST for point distributions with bounded expansion constant [88]. In our experiments, this algorithm turned out to be slower than GeoMST2, hence we did not include results from this work in our graphs.

Brennan [17] presented a modification to Kruskal’s classic minimum spanning tree algorithm [71] that operated in a manner similar to quicksort; splitting an edge set into “light” and “heavy” subsets. Recently, Osipov et al. [100] further expanded this idea by adding a multi-core friendly filtering step designed to eliminate edges that were obviously not in the MST (Filter-Kruskal). Currently, this algorithm seems to be the most practical algorithm for computing MSTs on multi-core machines.

The algorithm presented in this thesis uses well separated pair decomposition in combination with a modified version of Filter-Kruskal for computing GMSTs.

### 3.2 GeoFilterKruskal Algorithm

Our GeoFilterKruskal algorithm computes a GMST for $P \subseteq \mathbb{R}^d$. Kruskal’s [71] algorithm shows that given a set of edges, the MST can be constructed by considering edges in increasing order of weight. Using Lemma 2.1.1, the GMST can be computed by running Kruskal’s algorithm on the BCCP edges of the WSPD of $P$. When Kruskal’s algorithm adds a BCCP edge $(u, v)$ to the GMST, where $u, v \in P$, it uses the UnionFind data structure to check whether $u$ and $v$ belong to the same connected component. If they do, that edge is discarded. Otherwise, it is added to the GMST. Hence, before testing for an edge $(u, v)$ for inclusion into the GMST, it should always attempt to add all BCCP edges $(u', v')$ such that $\text{Dist}(u', v') < \text{Dist}(u, v)$. GeoMST2 [98] avoids the computation of BCCP for many well separated pairs that already belong to the same connected component. Our algorithm uses this crucial observation as well. Algorithm 2 describes the GeoFilterKruskal algorithm in more detail.
Algorithm 2 GeoFilterKruskal Algorithm

Require: $S = \{(a_1, b_1), \ldots, (a_m, b_m)\}$ is a WSPD, constructed from $P \subseteq \mathbb{R}^d$; $T = \{}$.
Ensure: BCCP Threshold $\beta \geq 6$.

procedure GeoFilterKruskal(Sequence of WSPs : $S$, Sequence of Edges : $T$, UnionFind : $G$, Integer : $\beta$)
\[
\begin{align*}
E_l &= E_u = E_{l1} = E_{l2} = \emptyset \\
&\text{for all } (a_i, b_i) \in S \text{ do} \\
&\quad \text{if } (|a_i| + |b_i|) \leq \beta \text{ then } E_l = E_l \cup \{(a_i, b_i)\} \text{ else } E_u = E_u \cup \{(a_i, b_i)\}
\end{align*}
\]
\[\rho = \min\{\text{MinDist}(a_i, b_i) : (a_i, b_i) \in E_u, i = 1, 2, \ldots, m\}\]
\[\text{for all } (a_i, b_i) \in E_l \text{ do} \\
&\quad \{u, v, \delta\} = \text{BCCP}(a_i, b_i) \\
&\quad \text{if } (\delta \leq \rho) \text{ then } E_{l1} = E_{l1} \cup \{(u, v)\} \text{ else } E_{l2} = E_{l2} \cup \{(u, v)\}
\]
Kruskal$(E_{l1}, T, G)$
$E_{new} = E_{l2} \cup E_u$
Filter$(E_{new}, G)$
\[\text{if } (|T| < (n-1)) \text{ then GeoFilterKruskal}(E_{new}, T, G, \beta + 1)\]
end procedure

procedure Kruskal(Sequence of WSPs : $E$, Sequence of Edges : $T$, UnionFind : $G$)
\[\text{Sort}(E)\text{: by increasing BCCP distance}\]
\[\text{for all } (u, v) \in E \text{ do} \\
&\quad \text{if } G.\text{Find}(u) \neq G.\text{Find}(v) \text{ then } T = T \cup \{(u, v)\}; \ G.\text{Union}(u, v); 
\]
end procedure

procedure Filter(Sequence of WSPs : $E$, UnionFind : $G$)
\[\text{for all } (a_i, b_i) \in E \text{ do} \\
&\quad \text{if } (G.\text{Find}(u) = G.\text{Find}(v) : u \in a_i, v \in b_i) \text{ then } E = E \setminus \{(a_i, b_i)\}
\]
end procedure

end procedure
The input to the algorithm is a WSPD of the point set $P \subseteq \mathbb{R}^d$. All procedural variables are assumed to be passed by reference. The cardinality of a WSP refers to the number of points in the entire pair. A WSP $(A, B)$ is connected if all points in $A$ and $B$ belong to the same connected component. The set of WSPs $S$ is partitioned into set $E_l$ that contains WSPs with cardinality less than $\beta$ (initially 2), and set $E_u = S \setminus E_l$. We then compute the BCCP of all elements of set $E_l$, and compute $\rho$ equal to the minimum $\text{MinDist}(a, b)$ for all $(a, b) \in E_u$. $E_l$ is further partitioned into $E_{l1}$, containing all elements with a BCCP distance less than $\rho$, and $E_{l2} = E_l \setminus E_{l1}$. $E_{l1}$ is passed to the KRSKAL procedure, and $E_{l2} \cup E_u$ is passed to the FILTER procedure. By using this sort based approach, we eliminate the need to maintain a priority queue [98] unlike GeoMST2. The KRSKAL procedure adds the edges to the GMST or discards them if they are connected. FILTER removes all connected WSPs. The GeoFILTERKRSKAL procedure is recursively called, increasing the threshold value ($\beta$) by one each time, on the WSPs that survive the FILTER procedure, until we have constructed the minimum spanning tree.

### 3.2.1 Correctness

Given previous work by Kruskal [36] as well as Callahan [18], it is sufficient to show two facts to ensure the correctness of our algorithm. First, we are considering WSPs to be added to the GMST in the order of their BCCP distance. This is obviously true considering WSPs are only passed to the KRSKAL procedure if their BCCP distance is less than the lower bound on the BCCP distance of the remaining WSPs. Second, we must show that the FILTER procedure does not remove WSPs that should be added to the GMST. Once again, it is clear that any edge removed by the FILTER procedure would have been removed by the KRSKAL procedure eventually, as they both use the UnionFind structure to determine connectivity.

### 3.2.2 Analysis of the running time

The real bottleneck of this algorithm, as well as the one proposed by Narasimhan [98], is the computation of the BCCP\(^1\). If $|A| = |B| = \mathcal{O}(n)$, the BCCP algorithm stated in section 3.2 has a worst case time complexity of $\mathcal{O}(n^2)$. Since we have to process $\mathcal{O}(n)$ edges, naively, the computation time for GMST will be $\mathcal{O}(n^3)$ in the worst case. In this section, we analyze the proof based on the construction of WSPD using quadtrees. The overall running time does not change when fair split tree is used.

**High Probability Bound Analysis.** In this section we show that algorithm 2 takes one sort plus $\mathcal{O}(n)$ additional steps, with high probability (WHP) [97], to com-

\(^1\)According to the algebraic decision tree model, the lower bound of the set intersection problem can be shown to be $\Omega(n \log n)$ [46]. We can solve the set intersection problem using BCCP. If the BCCP distance between two sets is zero, we can infer that the sets intersect, otherwise they do not. Since the set intersection problem is lower bounded by $\Omega(n \log n)$, the BCCP computation is also lower bounded by $\Omega(n \log n)$. 

26
pute the GMST. Let $Pr(\mathcal{E})$ denote the probability of occurrence of an event $\mathcal{E}$, where $\mathcal{E}$ is a function of $n$. An event $\mathcal{E}$ is said to occur WHP [97] if given $\mu > 1$,

$$Pr(\mathcal{E}) > 1 - 1/n^\mu. \quad (3.1)$$

Let $P$ be a set of $n$ points chosen uniformly from a unit hypercube $H$ in $\mathbb{R}^d$. Given this, we state the following lemma from [98]. We show that this lemma holds true if the above definition of WHP is used.

**Lemma 3.2.1** Let $C_1$ and $C_2$ be convex regions in $H$ such that $\nu \leq \text{Vol}(C_1)/\text{Vol}(C_2) \leq 1/\nu$ for some constant $0 < \nu < 1$. If $|C_1 \cap P|$ is bounded by a constant, then with high probability $|C_2 \cap P|$ is also bounded by a constant.

**Proof** Let $n_1 = |C_1 \cap P|$. Let $N_2$ be a random variable denoting the number of points in $((C_2 \setminus C_1) \cap P)$. According to the statement of the lemma, $n_1$ is $O(1)$. It has been shown in [98] that $Pr(N_2 = n_2) \leq O(n_2^{n_1})/(1 + \nu)^{n_2}$. This probability tends to be arbitrarily small as $n_2 \to \infty$. Thus $n_2$ cannot be arbitrarily high, which supports the claim of the lemma according to the definition of WHP used by Narasimhan et al. [98]. We now show that the lemma holds true if the definition of WHP given in equation (3.1) is applied to the proof.

To support our claim, we need to show that given a constant $c > 1$, $Pr(N_2 = n_2) \leq O(n_2^{n_1})/(1 + \nu)^{n_2} \leq 1/n_2^c$ i.e., $Pr(N_2 \neq n_2) > 1 - 1/n_2^c$, beyond some constant value of $n_2$. For a given $n_1$ and $c$, we show below that there exists an $n_0 \in \mathbb{N}$ ($\mathbb{N}$ is the set of all natural numbers), depending on $n_1$ and $c$, such that $n_2^{n_1}/(1 + \nu)^{n_2} \leq 1/n_2^c$, $\forall n_2 \geq n_0$. The inequality trivially holds for $n_2 = 1$. We will now see if it holds for all $n_2$ beyond some $n_0 \in \mathbb{N}$. We have

$$n_2^{n_1}/(1 + \nu)^{n_2} \leq 1/n_2^c \Rightarrow n_2^{(n_1+c)} \leq (1 + \nu)^{n_2} \quad (3.2)$$

Since $0 < \nu < 1$, $(1 + \nu) < 2$. Replacing $(n_1 + c)$ by $\xi$ in (3.2) we have

$$n_2^\xi < 2^{n_2} \quad (3.3)$$

Taking logarithm (base 2) on both sides and applying transposition to (3.3), we get

$$n_2/\log n_2 > \xi \quad (3.4)$$

$\xi = (n_1 + c)$ is a fixed constant and $n_2/(\log n_2)$ is an increasing function of $n_2$, $\forall n_2 \geq 3$. So for a given $\xi$, $\exists$ an $n_0 \in \mathbb{N}$, such that $\forall n_2 \geq n_0$, inequation (3.4) holds good. For example, if $n_1 = 1$ and $c = 2$, the inequality holds for $n_0 = 10$. Hence, given $c > 1$, $\forall n_2 \geq n_0 \in \mathbb{N}$, we have

$$Pr(N_2 \neq n_2) \geq (1 - 1/n_2^c) \quad (3.5)$$

In order to satisfy our claim, we need the r.h.s. in inequation (3.5) to be strictly greater than the l.h.s. Since $(1-1/n_2^c)$ is a strictly increasing function of $n_2$, $n_0' = n_0 + 1$ will satisfy inequation (3.5) with the '=' sign dropped. Thus the bounding constant of $n_2$ is $n_0'$. Hence $|C_2 \cap P|$ is bounded by a constant WHP.
In the remainder of the section, we fix the constant $\mu$ in the definition of WHP to 2.

**Lemma 3.2.2** Algorithm 2 filters out WSPs that have more than 6 points.

**Proof** The proof of this lemma is similar to the one for GeoMST2. Consider a WSP $(a, b)$. Let us assume, w.l.o.g., that $|a| \geq 7$. We will show that, in this case, we do not need to compute the BCCP distance of $(a, b)$ WHP. Let $\overrightarrow{pq}$ be a line segment joining $a$ and $b$ such that the length of $\overrightarrow{pq}$ (let us denote this by $|\overrightarrow{pq}|$) is MinDist$(a, b)$. Let $C_1$ be a hypersphere centered at the midpoint of $\overrightarrow{pq}$ and with radius $|\overrightarrow{pq}|/4$. Let $C_2$ be another hypersphere with the same center but radius $3|\overrightarrow{pq}|/2$. Since $a$ and $b$ are well separated, $C_2$ will contain both $a$ and $b$. Now, $\Vol(C_1)/\Vol(C_2) = 6^{-d}$. Since $C_1$ is a convex region, if $|C_1|$ is empty, then by Lemma 3.2.1, $|C_2|$ is bounded by a constant ($\gamma$, say) WHP. In fact, if we replace $n_1$ by 0 (since $|C_1| = 0$) in Lemma 3.2.1, then according to inequations 3.4 and 3.5, $\gamma = 6$. But $C_2$ contains $a$ which has at least 7 points. Hence $C_1$ cannot be empty WHP. Let $a \in a, b \in b$ and $c \in C_1$. Also, let the pair $(a, c)$ and $(b, c)$ belong to WSPs $(u_1, v_1)$ and $(u_2, v_2)$ respectively. Note that BCCP$(a, b)$ must be greater than BCCP$(u_1, v_1)$ and BCCP$(u_2, v_2)$.

Since our algorithm adds the BCCP edges by order of their increasing distance, $c$ and the points in $a$ will be connected before the BCCP edge between $a$ and $b$ is examined. The same is true for $c$ and the points in $b$. This causes $a$ and $b$ to belong to the same connected component WHP. If the number of points in any one of the quadtree boxes in $(u_1, v_1)$ or $(u_2, v_2)$ is 7 or more, then the existence of a point, WHP, in between $u_1$ and $v_1$, or between $u_2$ and $v_2$, can be shown by recursively arguing as above. WHP, there is no such point only when both the quadtree boxes in a WSP have at most 6 points. Our algorithm computes the BCCP of the WSPs in order of their increasing cardinality. So all the BCCP pairs of the WSPs of size at most 6 will be computed before computing the same for a pair like $(a, b)$. Once the pairs with less than 6 points are computed and passed to the Kruskal’s algorithm, the rest of the WSPs will be filtered out WHP. For example, if both $(u_1, v_1)$ and $(u_2, v_2)$ are of size $\leq 6$, then the BCCP of both these pairs will be computed before that of $(a, b)$. In addition, $(a, b)$ will be filtered out WHP after the BCCPs of $(u_1, v_1)$ and $(u_2, v_2)$ are passed to the Kruskal’s algorithm. Thus, our filtering step will get rid of the well separated pair $(a, b)$ before we need to compute its BCCP edge, WHP.

**Lemma 3.2.3** WHP, the total running time of the UnionFind operation is $O(\alpha(n)n)$.

**Proof** Lemma 3.2.2 shows that, WHP, we only need to compute BCCP distances of WSPs of constant size. Since we compute BCCP distances incrementally, WHP, the number of calls to the GeoFilterKruskal procedure is also bounded above by $O(1)$. In each of such calls, the Filter function is called once, which in turn calls the Find$(u)$ function of the UnionFind data structure $O(n)$ times. Hence, there are in total $O(n)$ Find$(u)$ operations done WHP. Thus the overall running time of the Union() and Find() operations is $O(\alpha(n)n)$ WHP (see the paragraph on UnionFind in Section 2.1.4).
Theorem 3.2.4 Algorithm 2 takes one sort plus $O(n)$ additional steps, WHP, to compute the GMST.

Proof We partition the list of well separated pairs twice in the GeoFilterKruskal method. The first time we do it based on the need to compute the BCCP of the well separated pair. We have the sets $E_l$ and $E_u$ in the process. This takes $O(n)$ time except for the BCCP computation. In $O(n)$ time we can find the pivot element of $E_u$ for the next partition. This partitioning also takes linear time. From Lemma 3.2.2, we can infer that the recursive call on GeoFilterKruskal is performed $O(1)$ times WHP. Thus the total time spent in partitioning is $O(n)$ WHP. Since the total number of BCCP edges required to compute the GMST is $O(n)$, by Lemma 3.2.2, the time spent in computing all such edges is $O(n)$ WHP. Total time spent in sorting the edges in the base case is $O(n \log n)$. Including the time to compute the Morton order sort for the WSPD, the total running time of the algorithm is one sort plus $O(n)$ additional steps WHP.

Remark 1. As explained in Section 2, the Morton order sort required to construct the compressed quadtree for the WSPD is $O(n)$ if the coordinates of the points in $P$ are of size $O(\log n)$ and the word size is $\Omega(\log n)$. This is also applicable in case of sorting the edges inside the Kruskal procedure of algorithm 2 if the BCCP distances can be represented in $O(\log n)$ bits and the word size of the machine is $O(\log n)$ [59]. Thus the whole algorithm runs in $O(n)$ time in this case, WHP. On the contrary, GeoMST2 will always take $O(n \log n)$ steps. This is because it adds edges to the GMST one at a time and before each addition it has to invoke an insertion and/or deletion procedure in a priority queue of WSPs [98]. Each such operation is $O(\log n)$ [36]. Since there are $O(n)$ WSPs, GeoMST2 will run in $O(n \log n)$ steps. It is to be noted that in case of a fair split tree based WSPD, our algorithm will also take an overall running time of $O(n \log n)$.

Remark 2. Using a $k$-nearest neighbor graph, we can modify algorithm 2 such that its running time is $O(n)$ WHP, if points in $P$ have integer coordinates. One can first compute the minimum spanning forest of the $k$-nearest neighbor graph of the point set, for some given constant $k$, using a randomized linear time algorithm [64]. In this forest, let $T'$ be the tree with the largest number of points. Rajasekaran [109] showed that there are only $n^\lambda$ points left to be added to $T'$ to get the GMST, where $\lambda \in (0, 1)$. In our algorithm, if the set $T$ is initialized to $T'$, then our analysis shows that WHP, only $O(n^\lambda)$ additional computations will be necessary to compute the GMST.

Remark 3. Computation of GMST from $k$-nearest neighbor graph can be parallelized efficiently in a CRCW PRAM. From our analysis we can infer that in case of a uniformly randomly distributed set of points $P$, if we extract the $O(1)$ nearest neighbors for each point in the set, then these edges will contain the GMST of $P$ WHP. Callahan [18] showed that it is possible to compute the $k$-nearest neighbors of all
Figure 3.1: This figure demonstrates the run time gains of the algorithm as more threads are used. We present scaling for two architectures. The AMD line was computed on the machine described in Section 4.8. The Intel line used a machine with four 2.66GHz Intel(R) Xeon(R) CPU X5355, 4096 KB of L1 cache, and 8GB total memory. For additional comparison, we include KNNG construction time using a parallel 8-nearest neighbor graph algorithm. All cases were run on 20 data sets from uniform random distribution of size $10^6$ points, final total run time is an average of the results.

points in $P$ in $O(\log n)$ time using $O(kn)$ processors. Hence, using $O(n)$ processors, the minimum spanning tree of $P$ can then be computed in $O(\log n)$ time [121].

**Remark 4.** We did not pursue implementing the algorithms described in remarks two and three because they are inherently Monte Carlo algorithms [95]. While they can achieve a solution that is correct with high probability, they do not guarantee a correct solution. One can design an exact $\text{GMST}$ algorithm using $k$-nearest neighbor graphs; however preliminary experiments using the best practical parallel nearest neighbor codes [33, 8] showed construction times that were slower than the GeoFilterKruskal algorithm. See Figure 3.1.

**Remark 5.** If sorting is allowed to be done in parallel, then Algorithm 2 takes $O(n)$ steps to compute the $\text{GMST}$, WHP irrespective of the assumption on the word-size of the machine as well as the coordinates of the input points. Given $n$ elements to sort and $p$ ($1 \leq p \leq n$) processors, each processor is assigned $O\left(\frac{n}{p}\right)$ elements. Then applying sequential mergesort to each processor, we have each of the $O\left(\frac{n}{p}\right)$ elements sorted in $O\left(\frac{n}{p} \log\left(\frac{n}{p}\right)\right)$ time. We will now merge $p$ sorted arrays, with two at a time parallely, thus taking $O(\log p)$ steps to produce the ultimate sorted array. The total
time \( (T_{\text{merge}}) \) for merging will then be

\[
T_{\text{merge}} = \left[ O\left(\frac{n}{p}\right) + O\left(\frac{2n}{p}\right) + \ldots + O(n) \right] \\
= \left[ \tau \frac{n}{p} (1 + 2 + 2^2 + \ldots + 2^{\log p}) \right] (\tau \in \mathbb{R}^+) \\
= \tau \frac{n}{p} (2p - 1)
\]

Thus \( T_{\text{merge}} \) is \( O(n) \). It now remains to calculate the value of \( p \) such that

\[ O\left(\frac{n}{p} \log \left(\frac{n}{p}\right)\right) = O\left(\frac{n}{p} \ln \left(\frac{n}{p}\right)\right) = O(n), \] where \( \ln n = \log_e n \).

We obtain the value of \( p \) as follows.

\[
\frac{n}{p} \ln \left(\frac{n}{p}\right) = n
\]

\[
\Rightarrow (p + \ln p) = \ln n \quad (3.6)
\]

Let \( y = \ln n \). Exponentiating both sides of equation 3.6,

\[ pe^p = e^y \quad (3.7)\]

Equation 3.7 is in the form of Lambert’s \( W \) function [35], where \( p = W(e^y) \). A nice property of the function [57] states that

\[ W(e^y) \leq [\ln(e^y) - \frac{1}{2} \ln \ln(e^y)] \quad (3.8)\]

which implies \( p = O(\ln n) = O(\log n) \) processors is sufficient to sort in linear time. Thus the total running time for \texttt{GeoFk1} is \( O(n) \) in this case.

\[ \text{3.2.3 Parallelization} \]

Parallelization of the \texttt{WSPD} algorithm. In our sequential version of the algorithm, each node of the compressed quadtree computes whether its children are well separated or not. If the children are not well separated, we divide the larger child node, and recurse. We parallelize this loop using \texttt{OpenMP} [39] with a dynamic load balancing scheme. Since for each node there are \( O(1) \) candidates to be well separated [25], and we are using dynamic load balancing, the total time taken in \text{CRCW PRAM}, given \( p \) processors, to execute this algorithm is \( O\left(\left\lceil (n \log n)/p \right\rceil + \log n \right) \) including the preprocessing time for the Morton order sort, if the \texttt{WSPD} is generated using a quadtree. Using \( O(n) \) processors, construction time for \texttt{FST} as well as \texttt{WSPD} improves to \( O(\log n) \) [121].
Parallelization of the GeoFilterKruskal algorithm. Although the whole of algorithm 2 is not parallelizable, we can parallelize most portions of the algorithm. The parallel partition algorithm [108] is used in order to divide the set \( S \) into subsets \( E_l \) and \( E_u \) (See Algorithm 2). \( \rho \) can be computed using parallel prefix computation. In our actual implementation, we found it to be more efficient to wrap it inside the parallel partition in the previous step, using the atomic compare-and-swap instruction. The further subdivision of \( E_l \), as well as the Filter procedure, are just instances of parallel partition. The sorting step used in the Kruskal procedure as well as the Morton order sort used in the construction of the compressed quadtree can also be parallelized [108]. We use OpenMP to do this in our implementation. Our efforts to parallelize the linear time quadtree construction showed that one can not use more number of processors on multi-core machines to speed up this construction, because it is memory bound.

Parallelization of GeoMST2. The WSPD construction can be parallelized using the same technique as explained earlier. But we did not implement a parallel version of GeoMST2 due to the following reason. One of the parameters to measure the performance of parallel priority queues is the access pattern of the priority queue [43]. Each thread puts an item with a certain priority into the queue or gets an item from the queue, and after that does some “local” work. The amount of local work determines the contention level and is used to simulate realistic application behavior. At maximum contention level no local work (“working delay”) is done. In the absence of any local work, parallel priority queues are less efficient than their sequential counterparts because the cost of concurrency control outweighs the benefits [43]. In case of GeoMST2, WSPs can be pushed into the priority queue parallelly and no local computation is required to be done by any thread, thus giving rise to the situation of high contention. Due to this reason we did not implement a parallel version of GeoMST2.

3.3 Experimental Setup

The GeoFilterKruskal algorithm was tested in practice against several other implementations of geometric minimum spanning tree algorithms. We chose a subset of the algorithms compared in [98], excluding some based on the availability of source code and the clear advantage shown by some algorithms in the aforementioned work. Based on the results in the previous section, we elected to implement the Geometric Filter Kruskal algorithm using fair split trees, instead of quadtrees. Table ?? lists the algorithms that will be referred to in the experimental section.

GeoFilterKruskal was written in C++ and compiled with g++4.3.2 with -O3 optimization. Parallel code was written using OpenMP [39] and the parallel mode extension to the STL [108]. C++ source code for GeoMST2 and Triangle were provided by Narasimhan. In addition, Triangle used Shewchuk’s triangle library for Delaunay triangulation [118]. The machine used has 8 Quad-Core AMD Opteron(tm) Processor
Table 3.1: Algorithm Descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeoFK#</td>
<td>Our implementation of Algorithm 2. There are two important differences between the implementation and the theoretical version. First, the BCCP Threshold $\beta$ in section 3.2 is incremented in steps of size $\mathcal{O}(1)$ instead of size 1, because this change does not affect our analysis but helps in practice. Second, for small well separated pairs (less than 32 total points) the BCCP is computed by a brute force algorithm. In the experimental results, GeoFk1 refers to the algorithm running with 1 thread. GeoFk8 refers to the algorithm using 8 threads. This implementation used a fair split tree, as opposed to the quadtree.</td>
</tr>
<tr>
<td>GeoMST</td>
<td>Described by Callahan and Kosaraju [18]. This algorithm computes a WSPD of the input data followed by the BCCP of every pair. It then runs Kruskal’s algorithm to find the MST.</td>
</tr>
<tr>
<td>GeoMST2</td>
<td>Described in [98]. This algorithm improves on GeoMST by using marginal distances and a priority queue to avoid many BCCP computations.</td>
</tr>
<tr>
<td>Triangle</td>
<td>This algorithm first computes the Delaunay Triangulation of the input data, then applies Kruskal’s algorithm. Triangle only works with two dimensional data.</td>
</tr>
</tbody>
</table>
Table 3.2: Point Distribution Info

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unif</td>
<td>$c_1$ to $c_d$ chosen from unit hypercube with uniform distribution ($U^d$)</td>
</tr>
<tr>
<td>annul</td>
<td>$c_1$ to $c_2$ chosen from unit circle with uniform distribution, $c_3...c_d$ chosen from $U^d$</td>
</tr>
<tr>
<td>arith</td>
<td>$c_1 = 0, 1, 4, 9, 16...$ $c_2$ to $c_d$ are 0</td>
</tr>
<tr>
<td>ball</td>
<td>$c_1$ to $c_d$ chosen from unit hypersphere with uniform distribution</td>
</tr>
<tr>
<td>clus</td>
<td>$c_1$ to $c_d$ chosen from $10$ clusters of normal distribution centered at $10$ points chosen from $U^d$</td>
</tr>
<tr>
<td>edge</td>
<td>$c_1$ chosen from $U^d$, $c_2$ to $c_d$ equal to $c_1$</td>
</tr>
<tr>
<td>diam</td>
<td>$c_1$ chosen from $U^d$, $c_2$ to $c_d$ are 0</td>
</tr>
<tr>
<td>corn</td>
<td>$c_1$ to $c_d$ chosen from $2^d$ unit hypercubes, each one centered at one of the $2^d$ corners of a $(0,2)$ hypercube</td>
</tr>
<tr>
<td>grid</td>
<td>$n$ points chosen uniformly at random from a grid with $1.3n$ points, the grid is housed in a unit hypercube</td>
</tr>
<tr>
<td>norm</td>
<td>$c_1$ to $c_d$ chosen from $(-1,1)$ with normal distribution</td>
</tr>
<tr>
<td>spok</td>
<td>For each dimension $d'$ in $d$ $\frac{n}{d'}$ points chosen with $c_{d'}$ chosen from $U^1$ and all others equal to $\frac{1}{2}$</td>
</tr>
</tbody>
</table>

Table 3.3: Surface Reconstruction Timings (Secs)

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Points</th>
<th>GeoMST2</th>
<th>GeoFk1</th>
<th>GeoFk8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puppy</td>
<td>500446</td>
<td>6.83</td>
<td>4.02</td>
<td>2.76</td>
</tr>
<tr>
<td>Jaguar</td>
<td>1070456</td>
<td>14.24</td>
<td>10.36</td>
<td>7.53</td>
</tr>
<tr>
<td>Davinci</td>
<td>327632</td>
<td>4.80</td>
<td>3.43</td>
<td>1.06</td>
</tr>
</tbody>
</table>

8378 with hyperthreading enabled. Each core has a L1 cache size of 512 KB, L2 of 2MB and L3 of 6MB with 128 GB total memory. The operating system was CentOS 5.3. All data was generated and stored as 64 bit C++ doubles.

In the next sections there are three distinct sets of graphs. The first set compares the performance of the fair split tree with the quadtree. The second graph displays total running time versus the dimensionality of the point set ranging from two to six dimensions. The points are obtained from uniform random distribution in a unit hypercube. The $L_2$ metric was used for distances in all cases, and all algorithms were run on the same random data set. Each algorithm was run on five data sets, and the results were averaged. Triangle was not used in this experiment because it is known to be inefficient for dimensions greater than two.

The third set of graphs shows the mean total running times for two dimensional data of various distributions, as well as the standard deviation. The distributions were taken from [98] (given $n$ $d$-dimensional points with coordinates $c_1...c_d$), shown in Table 3.2.

34
Figure 3.2: Results of comparisons of tree size between fair split trees and quad trees on the same set of data. Data was one million two dimensional points taken from uniform distribution.
Figure 3.3: Results of comparisons of construction time between fair split trees and quad trees on the same set of data. Data was one million two dimensional points taken from uniform distribution.
Figure 3.4: Results of comparisons of number of well separated pairs produced as the separation factor is varied. FST is being used for the purpose. Data was one million two dimensional points taken from uniform distribution.
Figure 3.5: Results of comparisons of the error in GMST length as the separation factor is varied. FST is being used for the purpose. Data was one million two dimensional points taken from uniform distribution.
3.4 Experimental Results

We first compare the performance of the fair split tree with the quadtree and then present our results on point sets from several distributions including real world data sets.

3.4.1 Comparing Quadtrees and Fair Split Trees

In this section we compare results of running our algorithm using the compressed quadtree versus the fair split tree. The data sets used were uniformly random, two dimensional points. Experiments run on data sets from other distributions and higher dimensions (up to 5) were not significantly different. Figure 3.2 shows the difference in the size of the WSPD set computed from the two trees. As expected, better clustering in the fair split tree yields fewer well separated pairs. Figure 3.3 shows the timing comparisons from using both of the trees. Even though the compressed quadtree can be constructed more quickly, the algorithm runs significantly faster given the fewer WSPs produced by the fair split tree.

It should be noted that these result invalidate some experimental results from the conference publication of this work, where an error in the separation factor for the well separated pair decomposition in the code produced fewer WSPs than required from the compressed quadtree. In that case, the MSTs produced had some small
Figure 3.7: Total running time for each algorithm for varying sized data sets of uniformly random points, as the dimension increases. Data sets ranged from $10^6$ to $10^7$ points for the 2-d data. For each data set size 5 tests were done and the results averaged.
Figure 3.8: Mean run time and standard deviation to compute the GMST on data sets of various distributions (see Table 3.2) using GeoMST and GeoFK1 in 2 dimensions. The data set size was $10^6$ points. For each data set size 5 tests were done and the results averaged.
Figure 3.9: Total running time for each algorithm for varying sized data sets of uniformly random points, as the dimension increases. Data sets ranged from $10^5$ to $10^6$ points for the 3-d data. For each data set size 5 tests were done and the results averaged.
Figure 3.10: Mean run time and standard deviation to compute the GMST on data sets of various distributions (see Table 3.2) using GeoMst2 and GeoFK1 in 2 dimensions. The data set size was $10^6$ points. For each data set size 5 tests were done and the results averaged.
Figure 3.11: Total running time for each algorithm for varying sized data sets of uniformly random points, as the dimension increases. Data sets ranged from $10^5$ to $10^6$ points for the 4-d data. For each data set size 5 tests were done and the results averaged.
Figure 3.12: Mean run time and standard deviation to compute the GMST on data sets of various distributions (see Table 3.2) using Triangle and GeoFK1 in 2 dimensions. The data set size was $10^6$ points. For each data set size 5 tests were done and the results averaged.
margin of error. Interestingly, it seems that if one accepts a small error in the MST, reducing the separation factor can produce approximate MSTs using fewer WSPs. Figures 3.5 and 3.4 show empirical results for this case. We varied the separation factor from $\sqrt{2}$ (the lowest factor that guarantees a correct result) down to $0.1 \ast \sqrt{2}$. At this time there are no theoretical guarantees on the error of the approximation. These results show that, in practice, it appears to be quite small.

### 3.4.2 Synthetic Data Sets

As shown in Figures 3.7, 3.8, 3.9, 3.10, 3.11 and 3.12, GeoFK1 performs favorably in practice for almost all cases compared to other algorithms (see Table 3.1). For each of our experiments, the performance of DualTreeBoruvka (DTB) [88] was slower than GeoMST2 or GeoFK1. This is the reason we left out results from DTB in our experiments. For example, in a 2-d dataset with 100,000 points from a uniform distribution, DTB took 37.3 seconds to compute the GMST, whereas each of GeoMST2 and GeoFK took around 0.7 seconds to complete the job. The relative difference of the computation times did not change by much for the other experiments. Hence we left out the performance graph of DTB from the graphs. In two dimensions, only Triangle outperforms GeoFK1. In higher dimensions, GeoFK1 is the clear winner when only one thread is used.

Each of GeoMST2 and GeoFK1 tends to slowdown as the dimension increases. See Figure 3.6. This is primarily because of the increase in the number of well separated pairs [98]. For example, the number of well separated pairs generated for a two dimensional uniformly distributed data set of size $10^6$ was approximately $10^7$, whereas a five dimensional data set of the same size had $10^8$ WSPs.

Figure 3.10 shows that in most cases, GeoFK1 performs better regardless of the distribution of the input point set. Apart from the fact that Triangle maintains its superiority in two dimensions (Figure 3.12), GeoFK1 performs better in all the distributions that we have considered, except when the points are drawn from arith distribution. In the data set from arith, the ordering of the WSPs based on the minimum distance is the same as based on the Bccp distance. Hence the second partitioning step in GeoFK1 acts as an overhead. The same experiments for data sets of other dimensions did not give significantly different results, and so were not included.

### 3.4.3 Real World Data Sets

Our experiments on real world data sets included 3-d point clouds obtained from Microsoft’s Kinect device and surface reconstruction data from the GAMMA team of INRIA-Paris. In case of Kinect, our data set consisted of 354 frames, with each frame having an average of 234225 points. However, for this data set we did not observe any noticeable difference in the performance of GeoMST2, GeoFK1 and GeoFK8. Nevertheless, using both GeoMST2 and GeoFK1, we were able to construct the GMSTs of
the frames at a rate of 3 per second with 16 threads, the job being scheduled using a standard producer-consumer routine. For the surface reconstruction data set, both the sequential and parallel version of our algorithm performed better than GeoMST2. Table 3.3 shows the results from three such examples. Figure 3.13 shows the models reconstructed by drawing the GMST of the data sets.

![Figure 3.13: Reconstructed models of a sleeping puppy, a jaguar head, and DaVinci’s head.](image)

### 3.5 Conclusion

This work strives to demonstrate a practical GMST algorithm, that is theoretically efficient on uniformly distributed point sets, works well on most distributions and is multi-core friendly. To that effect we introduced the GeoFILTERKruskal algorithm, an efficient, parallelizable GMST algorithm that in both theory and practice accomplished our goals. We proved a running time of \( O(n \log n) \), as well as provided extensive experimental results.
CHAPTER 4

INSTANT APPROXIMATE ONE CENTER ON ROAD NETWORKS VIA EMBEDDINGS

In this chapter, we introduce an approximate solution to the 1-center problem [28]. Using Euclidean embeddings, and reduction to fast nearest neighbor search, we devise the approximation scheme. On real world data sets, we conduct extensive experiments that indicate fast computation of constant factor approximate solutions for query sets much larger than previously computable using exact techniques. We provide a web interface for generic queries on a map interface.

**Our Contribution:** We summarize our contributions below.

- We propose a fast and approximate solution to the 1-center problem on the road network. To our knowledge, there does not exist any practical method for solving this problem instantly in the literature.

- We use embedding techniques to map the road network into low dimensional Euclidean space and empirically show that the Euclidean metric in the embedded space is a good approximation of the road network.

- We conduct extensive experiments on different data sets and show that embedding gives us fast and good approximate results.

- We provide a graphical interface for efficient access to our proposed algorithm for generic one-center queries.

The chapter is organized as follows. In the remainder of this section we define our problem formally and introduce the notation we use. Section 4.1 discusses background and related work, section 4.2 introduces our approximate algorithm. Sections 4.3 and 4.4 discuss the different error measures in our solution and the optimization criterion we used for embedding the road network graph in the Euclidean space. Section 4.5 justifies our optimization criterion. Sections 4.6 shows how we handle large data sets in the preprocessing phase. Section 4.7 describes the tools we use in the query phase.
In sections 4.8 and 4.9 we discuss our experimental setup and show our experimental results. Section 4.10 concludes the chapter.

4.1 Related Work

In many applications that manage spatial data (e.g., location-based services), the position and accessibility of spatial objects are constrained by spatial networks. In such cases, the actual distance between two objects corresponds to the length of the shortest path connecting them in the network. Recently, there has been an increasing interest in processing nearest neighbor queries over road networks [140]. Given a set $P$ of interesting objects (e.g., facilities) and a location $q$, the nearest neighbor query returns the nearest object of $q$ in $P$. Formally, the query retrieves a point $p \in P$, such that $d(p,q) \leq d(p',q) \forall p' \in P$, where $d()$ is a distance function (i.e., the network distance in our setting). The Aggregate Nearest Neighbor (ANN) problem introduced in section 1.2 is a generalization of the NN-search. In context of the Euclidean space, this problem is referred to as Group Enclosing Query, thus in this thesis we call it the Network Group Enclosing Query problem. Given a set $P$ of interesting objects, a set $Q$ of query points, and an aggregate function $f$ (e.g., sum, max) an ANN query retrieves the object $p$ in $P$, such that $f\{ (p,q_i) , \forall q_i \in Q \}$ is minimized. Such queries could be generally interpreted as a minimizing optimization problem defined on the nearest neighbor search. For example, when the aggregate operator is the SUM, the aggregate nearest neighbor query is equivalent to the group nearest neighbor query [101] where the goal is to find a point from $P$ that has the minimum SUM distance to all query points in $Q$. In the next section we digress from the network domain to the Euclidean domain, as it is easier to comprehend the problem when the distance metric is Euclidean. Following our discussion in the domain of Euclidean space, we will introduce the Network 1-center. This thesis attempts to solve the Network 1-center problem by using the solutions in the Euclidean domain.

4.1.1 The Euclidean Group Enclosing Query Problem

Given a set of points $P$ and a query set $Q$, a group enclosing query (GEQ) fetches the point $p^* \in P$ such that the maximum distance of $p^*$ to all points in $Q$ is minimized. This problem is equivalent to the aggregate nearest neighbor queries in spatial databases [102].

Minimum Bounding Box Method:. The state of the art method for the GEQ problem is the Minimum Bounding Box Method (MBM) from [102]. The principal methodology adapted by the MBM is the triangle inequality. It is a heuristic method that has $O(N+M)$ query time, where $N$ and $M$ are the sizes of $P$ and $Q$ respectively. It is an R-tree [55] based approach. An R-tree is an extension of B-tree [12] to higher dimensions. Points are grouped into Minimum Bounding Rectangles (MBR) which are recursively grouped into MBRs in higher levels of the tree. The grouping is based on
locality and bounded by page size. The MBM method recursively visits the nodes of the R-tree and prunes results by sequentially using two pruning conditions. Let $\mathfrak{M}$ be the MBR of the query group $Q$, $\mathfrak{M}_i$ be the current node (and its MBR) from the R-tree, and $\text{bestDist}$ be the enclosing distance of the best max-ANN found so far. The two pruning conditions are $\minDist(\mathfrak{M}, \mathfrak{M}_i) \geq \text{bestDist}$, and $\max_{q_i \in Q}(\minDist(\mathfrak{M}, q_i)) \geq \text{bestDist}$, where $\minDist(A, B)$ is the minimum distance between $A$ and $B$, $A$ and $B$ are either the nodes of the R-tree or a point. Either condition can safely prune the node $\mathfrak{M}_i$ (and its subtree). The second condition is applied only for nodes that pass the first one in order to save unnecessary computations. However, the MBM method only explores the triangle inequality and important pruning and optimization opportunities are missed [79]. The algorithm described in the next section attempts to alleviate this problem, thus designing a more efficient method for exact answers to the GEQ problem.

GEQs Algorithm: The GEQs algorithm [79] uses three types of pruning strategies. The first observation is that the furthest point in $P$ from the query set $Q$ is determined by the points on the convex hull [41] of $Q$ [79]. So it is sufficient to prune out the points which are not on the convex hull, although in the worst case, all the points in $Q$ can constitute the vertices of the convex hull. The second observation is based on the minimum enclosing ball (MEB($Q$)) [41] of $Q$. Let $B(c, r)$ be the ball with center $c$ and radius $r$, $nn(c, P)$ be the nearest neighbor of $c$ in $P$, $nn(c, P) = p$, and the distance between $p$ and $c$ be $\lambda$. Then the following lemma holds. Please refer to [79] for proof.

Lemma 4.1.1 If $p^*$ is the optimal answer to the GEQ problem, then $p^* \subseteq B(c, r + \lambda)$ [79].

The third stage of pruning, done by using furthest point voronoi diagrams [41, 79] help calculate the minimum and maximum distances of an MBR in the R-tree to a query group $Q$. Given an R-tree node’s MBR $u$ and a query group $Q$, the tight minimal and maximal enclosing distances for all possible points inside $u$ clearly help us decide whether we can prune $u$ or not. There are infinite number of possible points that $u$ may contain (even if there are only finite number of actual points from $P$ in $u$, their precise locations are unknown given only the MBR $u$). Hence, infinite number of possible enclosing distances exist. We are interested at efficiently finding out both the minimum and the maximum possible enclosing distances for an MBR node $u$ w.r.t a query group $Q$, as these values tell us what the best and the worst possible scenarios are for any point inside $u$ to enclose $Q$. Let $\minEdist(u, Q) = \min_{p \subseteq u} r_{p,Q}$, $\maxEdist(u, Q) = \max_{p \subseteq u} r_{p,Q}$, where $r_{p,Q}$ is the radius of MEB($Q$) with center at $p$. If one can efficiently compute $\minEdist(u)$ and $\maxEdist(u)$, a pruning method based on an R-tree with the branch and bound principle is immediately possible. The key observation is that given two MBRs $u_1$ and $u_2$, if $u_2$’s $\minEdist$ is larger than $u_1$’s $\maxEdist$, we can safely prune the entire subtree rooted at $u_2$. To compute them (minEdist and maxEdist) efficiently, we need the help of the furthest voronoi cells
(FVCs, same as the furthest voronoi diagram) [79]. The FVCs of Q cover the entire space [79]. Let us define FC(q) for a point q as follows [79]: any point contained by FC(q) has q as its furthest neighbor from the point set Q. The FVCs of Q is the collection of furthest cells for all points in Q. That being said, for any MBR u, there will be some qᵢ’s from Q such that FC(qᵢ)s intersect with (or only one FC(qᵢ) fully contains) u [79]. Li et al. [79] show that these qᵢ’s must be some vertices from the convex hull of Q. Furthermore, the union of these intersections will be equal to u. Using these observations, the R-tree can be pruned by compute maxEdist and minEdist efficiently. See [79] for details.

The exact search algorithm GEQs combines the three pruning techniques. Given Q and the R-tree on P, we first compute the convex hull (C_Q) to reduce the size of the query group. Then MEB(C_Q), the nearest neighbor of MEB(C_Q)’s center from P (with the help of the R-tree) and the furthest voronoi cells of C_Q are computed [9, 79]. Next, a priority queue L is maintained for all MBRs (points are treated as MBRs as well) that the algorithm has to search for. Entries in L are ordered by their minEdist distances. An R-tree node could be safely pruned without exploring its children nodes if either it does not intersect with B(c, r+λ) using range queries [41], or its minEdist is larger than the smallest maxEdist of any existing entries from the queue L. Initializing the queue with the root node, we repeatedly pop the head of the queue and insert its children nodes back to the queue if they cannot be pruned. The algorithm terminates when the head of the queue becomes a point.

Approximate Query Algorithms: The nearest neighbor search is a special case for the GEQ problem. Hence, any exact method for the GEQ problem will suffer from the curse of the dimensionality. The best hope for answering a GEQ query instance for data sets in high dimensions is to look for efficient and effective approximation algorithms, just as what people have done for the approximate nearest neighbor search in high dimensions [9, 79]. Li et al. [79] show an efficient way to compute √2-approximate solution to the GEQ problem in any fixed dimension using a nearest neighbor query. This answer is then used to obtain an optimal (1+ε)-approximate solution to the GEQ problem in fixed dimensions for any ε > 0, and extend it to high dimensions. The √2 approximation has been shown well to work in practice, never exceeding 1.05 for dimensions upto six. The (1+ε)-approximate answer is only of theoretical interest, because it involves constants that are exponential in the dimension. A followup on this problem where the GEQ answer is sought for a subset of the query points instead of the entire set, has been recently studied by Li et al. [81].

4.1.2 Network Group Enclosing Query

Papadias et al. [102] have given a complete, thorough treatment to a number of aggregate nearest neighbor queries in spatial databases. They develop an algorithm for ANN queries, utilizing connectivity information and spatial locality. They propose an algorithm using R-trees to efficiently compute Euclidean distance bounds and
eventually a method for answering ANNs queries using Euclidean distance bounds as lower bounds for network distances. ANNs queries are solely dependent on nearest neighbor search, so extensive work has been done in literature to optimize the search procedures in the network domain. Shahabi et al. [114] transform the spatial network to a high dimensional space and use simple distance functions to approximate the network distance but experiments were restricted to small sized road networks. Jensen et al. [61] discuss nearest neighbor queries for objects moving in a network. Shekhar and Yoo [117] study the problem of finding nearest neighbors along a given route instead of finding nearest neighbors from a single query point. Huang et al. [58] propose methods for solving shortest path queries with spatial constraints. Luo et al. [83] propose projection based filtering where they efficiently prune the data points without indexing. Instance optimal query processing have been proposed by Deng et al. [42]. A fast object search technique exploiting search space pruning has been proposed by Lee et al. [77]. Xu and Jacobsen [136] study proximity relations in road networks and achieve significant practical results. The 1-center problem can be solved by using any of these techniques. In this thesis, we aim to provide a new insight into this problem, by capturing the underlying geometry of the network and thus use existing tools in the computational geometry literature to come up with a reasonable approximate solution.

### 4.1.3 Exact Solution

The exact solution to the 1-center problem was proposed as the KH algorithm [65]. In the road network graph $G = (V, E)$, consider a set of query points $Q = \{q_1, q_2, ..., q_m\}$ lying on the road network and an edge $e = (v_1, v_2) \in E$, where $v_1, v_2 \in V$. For each of the two endpoints $v_1$ and $v_2$ in $e$, the algorithm computes the shortest path distance from $v_1$ and $v_2$ to $q_i \in Q, \forall i = 1, 2, ..., n$. It then plots the shortest path distance along the edge $e$ from $q_i$. The shortest path distance plot along $e$ from $q_i$ may be monotonically increasing, monotonically decreasing or both. See Figure 4.1. Each
of the plots is thus either a straight line or a union of two straight lines. Then the lowest point in the upper envelope of the intersection of these straight lines and the corresponding location in the edge is obtained. The distance of this location to the corresponding query point is recorded. The process is repeated for every edge and the location that gives the minimum of all the recorded distances is the answer to the 1-center query. This algorithm, although an exact one is not practical to use in large road networks because it involves pairwise shortest path computations for each pairwise distance. In our actual implementation, we optimize this procedure by computing the shortest paths from the query points only, instead of from each of the endpoints of every edge.

4.2 The Approximate Solution

We present a brief overview of our algorithm which is extremely simple and has two phases, namely the preprocessing phase and the query phase. The preprocessing phase, in turn is divided into two parts, the “Graph Generation Phase” and the “Embedding Phase”.

**Graph Generation Phase:** We obtain our datasets from OpenStreetMap website (http://cloudmade.com). The next task is to parse the map files to generate the road network graph, followed by calculation of the pairwise shortest path distances from each vertex. The distance matrix thus generated is fed into the embedding algorithm for mapping the points into an Euclidean space of a chosen dimension.

However, an actual query might come from anywhere on an edge of the graph. The particular point on the edge from where the query is coming from has no representation in the embedding. This is because that point is not a vertex in the graph and thus was not a member of the distance matrix. This problem can be fixed by using the following technique. Using the longitude and latitude of each vertex, its 2-dimensional coordinates are calculated. Each edge is divided into shorter, unit length edges using Steiner points [41]. We calculate the map coordinates (latitude,longitude) of a Steiner point by interpolation from the coordinates of original vertices, and also store the fraction (λ, say) that the Steiner point is along the edge. Finally, we store all vertices and Steiner points in a 2-d nearest neighbor (NN) [34] query structure. We denote this set of 2-d points, consisting of the vertices and the Steiner points, by \( V_2 \).

To handle a query, we use the NN structure to find the closest vertex or Steiner point, and then obtain the corresponding location in the embedded space. For a Steiner point, one can use \( \lambda \) to interpolate between the embedded positions of the endpoints of its edge.

In our actual implementation, we remove the nodes with degree 2 to compute the distance matrix for the embedding. Removing these nodes does not affect the shortest path calculations but reduces the data size. We keep the degree-2 nodes for constructing the 2-d NN structure. The number of these nodes on each edge are sufficient enough to act as our Steiner points.
Embedding Phase: In the embedding phase, we embed the graph in a low dimensional Euclidean space using Multidimensional Scaling (MDS) [22, 47, 78, 96, 105, 129, 133]. We denote the set of points in the embedded space by \( V_{de} \), where \( d_e \) is the dimension of the embedded space. After the embedding is done, we compute another \( \mathbf{NN} \) structure, this time using the points in \( V_{de} \). In Section 4.5, we experimentally determine the value of \( d_e \).

Query Phase: The input to the query phase are two \( \mathbf{NN} \) structures (one from \( V_2 \) and the other from \( V_{de} \)), the query points and a set \( Q_{de} \). Initially, \( Q_{de} = \emptyset \). The algorithm executes the following steps in this phase.

- **Step 1**: Find the \( \mathbf{NN} \) of each query point (pairs of longitude, latitude) from \( V_2 \) using the 2-d \( \mathbf{NN} \) data structure.

- **Step 2**: For each query point, if the \( \mathbf{NN} \) in \( V_2 \) has a representation \( q \in V_{de} \), add \( q \) to \( Q_{de} \). If the \( \mathbf{NN} \) is a Steiner point, interpolate its representation in \( \mathbb{E}^{de} \) and add the result to \( Q_{de} \).

- **Step 3**: Compute the minimum enclosing hypersphere of the points in \( Q_{de} \).

- **Step 4**: Find the nearest neighbor of the center of the minimum enclosing hypersphere from the points in \( V_{de} \). Report the nearest neighbor’s corresponding vertex in the original graph as the answer.

### 4.3 Measures of Distortion

**Measure of a Good Embedding**: The objective of a good embedding is to preserve the pairwise shortest path distances in the embedded space as closely as possible. Any measure to judge the quality of an embedding should be able to reflect this objective. In this chapter we use two types of measures: a scaled measure and an absolute ratio measure.

**Scaled Measure**: This measure uses a scaling factor to compute the distortion. Given \( D \geq 1 \), if there exists a number \( r > 0 \), such that

\[
r.d_{ij} \leq d'_{ij} \leq D.r.d_{ij}, \quad i, j = 1, \ldots, n.
\]

then the embedding is defined to be a \( D - \) embedding [89]. The infimum of the numbers \( D \) such that the inequality holds is called the distortion of the embedding. We pick \( r \) to be the minimum over all \( d'_{ij}/d_{ij} \), \( i, j = 1, \ldots, n \). The distortion is thus the maximum over all \( d'_{ij}/(d_{ij}r) \). We also compute \( d'_{ij}/(d_{ij}r) \) for each \( (i, j) \) pair and
refer to the average of the numbers obtained as “mean distortion”. A good embedding should have both the distortion and its mean to be as low as possible.

**Absolute Ratio Measure:** Apart from the above measure which involves scaling the distances by \( r \), we define another function which measures the difference between the absolute values of the pairwise distances obtained from the actual graph and the embedding. This helps us in evaluating the embedding from a set of pairwise distances sampled from the graph, especially when the size of the graph is too big to compute the actual distortion. We call this measure “AbsR” (abbreviation for absolute ratio) and define it as follows.

\[
\text{AbsR} = \max \left( \frac{d_{ij}}{d'_{ij}}, \frac{d'_{ij}}{d_{ij}} \right), \quad i, j = 1, 2, ..., n.
\]  

(4.2)

Unless otherwise mentioned, we use the scaled measure for calculating distortions. We now introduce our choice of MDS criterion.

### 4.4 Sammon Optimization Criterion

We sought an optimization criterion that will minimize both the distortion and its mean. We first introduce the optimization criterion we used in this work, and then emperically justify our choice in the following sections. We used the “sammon projection” [128] for our embedding purpose. The function is given by

\[
S = \frac{1}{\sum_{i<j} d_{ij}} \sum_{i<j} \frac{(d_{ij} - d'_{ij})^2}{d_{ij}}
\]  

(4.3)

The Sammon cost function differs from the classical MDS in the sense that it puts more emphasis on retaining distances that were originally small. The minimization of the stress function can be performed using various methods, such as the eigen-decomposition of a pairwise distance matrix, the conjugate gradient method, or a pseudo-Newton method [80]. We use the conjugate gradient method implemented in MATLAB.

### 4.5 Embedding Phase

In this section, we emperically establish the reason for our choice of the MDS optimization criterion by comparing it against the classical method and the PC algorithm. We also show how an optimal embedding can be obtained for our measure of distortion and mention the limitations in obtaining optimality for our data sets of interest.
Classical vs. Sammon based MDS. Since our pre-processing is mainly based on MDS, we first experimentally settled on the optimization criterion for MDS. We used MATLAB 7.11.0.584 for this purpose. We saw that the distortion of classical MDS (also referred to as Strain criterion) was orders of magnitude larger than Sammon. The mean distortion was also large for Strain compared to Sammon, as Figure 4.2 suggests. One advantage that Strain has compared to Sammon is speed (See Figure 4.3), but since this was pre-processing, and our main aim was to build better embeddings, the choice was clear.\(^1\)

![Figure 4.2: Mean distortion obtained for the Sammon and Strain criteria on Tallahassee road network data. We report the distortion only for Sammon, because the same for Strain was larger than 10^3. The data set has 1597 nodes and 2425 edges.](image)

From Figures 4.2 and 4.3, it was clear that not only was the distortion low when we went to 6 dimensions, the computation time for Sammon was also not very high. Hence in all our experiments, the embedding dimension was fixed to 6 whenever we used the Sammon projection. We repeated these experiments on several other road networks, but the results obtained were similar. Hence we do not report them here.

Sammon vs Place Center MDS. The place center (PC) algorithm [2] is a very recent embedding technique. It is a unified algorithmic framework for solving many variants of MDS, although Sammon function does not fall into the family of error functions that PC optimizes. The approach is based on an iterative local improvement method. The algorithm picks a point and moves it so that the cost function is locally

\(^1\)Two other optimization criteria for non-classical MDS, namely the “metricstress” and the “metricsstress” are available in MATLAB. In our experiments, these functions produced slightly worse results than Sammon as far as distortion and mean distortion are concerned. Also, speed of convergence was similar to that of Sammon. Hence we did not pick either of these two functions.
optimal and repeats this process until convergence. We ran PC on five different subgraphs from the map of USA. The dimension of embedding used here is 6. Our experiments showed that the distortions of the embeddings produced by PC did not change by much after the 5th dimension and continued to be worse than Sammon. In Figures 4.4 and 4.5 we present our results obtained by optimizing the fMDS cost function [2]. Our observations were similar when rMDS [2] and r²MDS [2] were optimized. From the figures, it is clear that Sammon is a better choice.

**Optimal Sum Error Matrix Embedding for Road Network Graph.** In this section we show how to produce an optimal embedding for our measure of distortion using a semidefinite program (SDP) and then state its practical limitations. The program follows:

\[
P1 : \text{Minimize } \sum_{ij} \epsilon'_{ij}
\]
\[
\text{s.t. } (1 - \epsilon'_{ij})d_{ij} \leq ||x_i - x_j|| \leq (1 + \epsilon'_{ij})d_{ij};
\]
\[
\epsilon'_{ij} \geq 0; \ i, j = 1, ..., n.
\]

To feed the program to a SDP solver, we modify it as follows. Squaring and expanding the first constraint gives us:

\[
(1 - \epsilon'_{ij})^2d_{ij}^2 \leq (x_i'x_i' - 2x_i'x_j + x_j'x_j') \leq (1 + \epsilon'_{ij})^2d_{ij}^2.
\]
Replacing \((1 - \epsilon_i'j)^2\) by \((1 - \epsilon_{ij})\), our final form of the semidefinite program becomes

\[
P2 : \text{Minimize} \sum_{ij} \epsilon_{ij}
\]
\[
\text{s.t.} \quad (1 - \epsilon_{ij})d_{ij}^2 \leq (y_{ii} - 2y_{ij} + y_{jj}) \leq (1 + \epsilon_{ij})d_{ij}^2;
\]
\[
\epsilon_{ij} \geq 0;
\]
\[
y_{ij} \geq 0; \quad i, j = 1, \ldots, n.
\]

In the above program \(\epsilon_{ij}\) and \(y_{ij}\) are \((i, j)\)th elements of the error matrix \(\epsilon\) and the Gram matrix \(Y\) of the embedded points, respectively. We used \text{SDPT-3} \[126, 127\] to
solve P2. However, as observed by [20], general purpose SDP-solvers are too memory intensive for even modestly sized programs. For this reason, in these experiments, we had to restrict our data set to \( n \leq 100 \) points. The same reason applies why we stuck to Sammon criterion for embedding road networks of larger sizes. The aim of this section is to show that we perform reasonably well when compared with the optimal result. We compare the performance of the Sammon function with that of SDP in Figure 4.6 and 4.7 on five different subgraphs chosen randomly from the map of USA. It is to be noted that SDPT-3 embeds in \( n \) dimensions while Sammon embeds in 6 dimensions. Reducing the dimension of the embedding by SDPT-3 will reduce the gap between the methods in the graphs. The abbreviations on the x-axes of the graphs show the names of the states from which the subgraphs came from.

![Figure 4.6: Comparison of distortions from SDPT-3 and Sammon.](image)

**Non Embeddable Metric Spaces:** There exists simple examples of metric spaces which are not isometrically embeddable into any Euclidean space [89]. Two such examples, given in Figure 8 are often encountered as subgraphs in road networks. We used our SDP as well as the Sammon projection to embed these graphs. Both the methods produced the exact graphs in the 2-dimensional Euclidean space, thus amounting to the same distortion.

**SDP with k-Nearest Neighbors.** Techniques like LLE [111] or IsoMap [125, 144] try to embed high dimensional data to low dimensions using information from the k-nearest neighbors (k-NN) of each data point. Empirically we tried to find a k such that using the pairwise shortest path distances of only the k-nearest neighbors of each point, we will be able to get an embedding with minimum distortion. For this
Figure 4.7: Comparison of mean distortions from SDPT-3 and Sammon.

Figure 4.8: Two examples of 4 point metric spaces that cannot be embedded isometrically into any Euclidean space. Each edge in both the graphs is of unit length.

For each point we restricted SDPT-3 to check for $nk$ constraints, instead of $n^2$. Using this SDP, we embedded the subgraphs used in the previous section again. For each $k$, we collected the distortion obtained. We present one of our observations in Figure 4.9. In the graph, we see that the smallest $k$ for which the distortion is lowest, is very close to $n$. Similar trend was observed for mean distortion. For each of the subgraphs that we embedded, the observations were similar. Due to this reason, we did not pursue the route of embedding our graphs using k-NN based methods like LLE or IsoMap.
4.6 Handling Large Data Sets

For large data sets, the MDS procedure becomes extremely memory intensive, because of the number of variables the function has to optimize. Several approximations to classical MDS have been proposed to address its poor scalability. One class of algorithms is based on a spring-mass model [22, 96]. [133] have suggested an improvement to Chalmers’ approach, and added user control to speed up performance. A disadvantage of spring-based models, in general, is that they are subject to local minima, and that they require an a priori assumption of the dataset’s underlying dimensionality [137]. FastMap [47], MetricMap [129], and Landmark MDS (LMDS) [78] approximate classical MDS by solving MDS for a subset of the data and fit the remainder to the solution. [105] shows how all three algorithms belong to a class of methods called Nyström algorithms, which approximates the solution to the eigenproblem of a large matrix and concludes that LMDS is the fastest and most accurate of the three. One disadvantage we face when using these algorithms for our purpose is that all these techniques require the entire distance matrix to be supplied as one of their inputs. Generating the distance matrix is expensive both in terms of computation and storage, especially in our case where the number of vertices in the road network graph goes up to roughly 27 million for the entire map of USA. Even if one generates the whole matrix and applies the exact algorithm to find the 1-center, it will involve \(O(mk)\) memory lookups during the query time, where \(m\) is the number of edges in the graph and \(k\) is the number of query points. Clearly, this can be expensive if \(m\) and/or \(k\) is very high.

In this work, we use a sampling based MDS (we will refer to it as FastMDS) proposed by [137]. In the next section we first give an overview of FastMDS, followed by our algorithm for handling large data sets.
Sampling Based FastMDS. The FastMDS approach is based on the observation that a submatrix along the diagonal of a dissimilarity matrix is itself a dissimilarity matrix. Instead of running MDS on the full $n \times n$ matrix $M$, it is partitioned along the diagonal into $p$ submatrices $M_1, M_2, ..., M_p$, each of size $n_p \times n_p$. We will refer to a submatrix as $M_i$, where $1 \leq i \leq p$. The MDS solution is then computed for each submatrix $M_i$. These individual MDS solutions are then stitched together by sampling $s$ points from each submatrix $M_i$ and putting them into an alignment matrix $N_{align}$ of size $sp \times sp$. In principle, $s$ should be at least $1+$ the estimated dimensionality of the dataset. In practice, $s$ is oversampled by a factor of 2 or more, to ensure that the data’s inherent dimensionality is captured. MDS is then run on $N_{align}$ to get $d_{low}$-dimensional coordinates for the sampled points. We now have two MDS solutions for each of the sampled points; one from performing MDS on $M_i$ and one from performing MDS on $N_{align}$. The next step is to compute an affine mapping between these two sets of solutions to align them up in a common coordinate system.

This is a linear least squares problem:

$$E_{N_{align}} = U \ast E_{M_i} + V$$

where $E_{M_i}$ is the MDS solution for the sample points from $M_i$ and $E_{N_{align}}$ is the solution from $N_{align}$ that corresponds to sampled points from $M_i$. Solving for $U$ and $V$ gives us a mapping between $M_i$ and $N_{align}$, which is then applied to the rest of $M_i$ to get $d_{low}$-dimensional coordinates for all $n/p$ points. The process of partitioning the submatrices is applied recursively, until the size of each submatrix is sufficiently low to run MDS on. The stopping condition is found as follows. Let $l \times l$ be the largest matrix that allows MDS to be executed efficiently. There are two issues that impact the performance of FastMDS on an $n \times n$ matrix. (i) the size of $M_i$ after subdivision and (ii) $p$, the number of submatrices that are to be stitched together at each conquer step. Ideally, the size of each submatrix after division should be as large as possible without exceeding $l \times l$. By the same token, the size of the alignment matrix $N_{align}$ should also be bounded by $l \times l$. The number of submatrices to be stitched together, $p$, should be the largest number such that $sp \leq l$.

Assuming $M$ can be embedded in $d_{low}$ dimensions, regular MDS guarantees an optimal embedding in $\mathbb{R}^{d_{low}}$ space. So there exists a $d_{low}$-dimensional coordinate for each point represented in $M$. If enough points are sampled from each submatrix $M_i$, the subset of points in $N_{align}$ should be able to capture the structure in the original point set. The running time for the entire algorithm is $O(n \log n)$ [137] where $n$ is the number of points in the input. Below we present our modified algorithm for handling large data.

### 4.6.1 Modified algorithm for Large Data Sets

It is clear from the overview in Section 4.6 that FastMDS does not use information about the pairwise shortest path distance between two points if they are in different
partitions unlike algorithms like LMDS, which requires the pairwise shortest path distance between non-landmark points [78] to produce the embedding. That being said, we present our algorithm for the preprocessing phase for generating the embedding formally. The method for the query phase does not get affected by large data sets.

The input to the preprocessing phase is the road network map and two scalars: $\mu$ and $\nu$. The following steps create a partition tree of the graph with at most $\mu$ nodes at each level.

**Step 1**: Generate the graphs (with and without the degree 2 nodes) from the road network. Generate the 2-d NN structure using the one having degree 2 nodes. Perform the following steps with the other graph.

**Step 2**: Binary Partition: Select a random node, and find the node furthest from it. This is the first node. Then select the furthest node from that node. This is the second node. For each other node, if it is closer to the first node, put it in the first partition. Otherwise, it is closer to the second node, and put it in the second partition.

**Step 3**: Repeat **Step 2** on the largest resulting subpartition until the graph is broken into $\mu$ subpartitions, or each partition is less than or equal to size $\nu$. A random sample of size $\frac{\nu}{\mu}$ is taken from each subpartition for later use.

**Step 4**: Repeat **Step 3** on each of the $\mu$ resulting subpartitions until all the subpartitions are of size less than or equal to $\nu$.

**Step 5**: Embed each subpartition and each sample partition using MDS with the Sammon optimization criterion.

**Step 6**: Use FastMDS to embed the original graph.

For large graphs, we found that $\mu = 20$ and $\nu = 1500$ worked well. For graphs smaller than 3000 nodes, we just ran MDS to compute the embedding.

### 4.7 Query Phase

We explain the steps in the query phase in this section. The first and the fourth steps involve NN computation. One of the most successful methods for Euclidean NN-search is to partition the entire point set using a tree type data structure. Queries are conducted by dropping a search ball on the tree and conducting searches of those nodes which intersect the ball. Tree based nearest neighbor searches have been found to work very well in practice in low dimensions; expected running time can be found to be $O(\log n)$ for $(1 + \epsilon)$-approximate solutions [9, 23]. We used a z-order based quadtree nearest neighbor search algorithm for implementing our algorithm [34].
The center of the minimum enclosing ball of the query points that is computed in step 3 of the query phase, is essentially the 1-center of points in the Euclidean space. Let \( Q \) be a set of points in \( \mathbb{R}^d \). Let \( B(c, r) \) denote a ball centered at \( c \) with radius \( r \). We use \( \text{MEB}(Q) \) to denote the minimum enclosing ball for \( Q \). \( \text{MEB}(Q) \) is the smallest (measured by \( r \)) ball \( B(c, r) \) such that \( Q \subseteq B(c, r) \). In this context, the operator \( \subseteq \) refers to the geometric containment, i.e., every point in \( Q \) is fully contained by the ball defined by \( B(c, r) \). If the center is fixed at \( p \), \( \text{MEB}(p, Q) \) denotes the smallest ball centered at \( p \) that encloses all points in \( Q \). Efficient algorithms for computing both exact and approximate \( \text{MEB} \)s are available in literature [92, 73, 138].

Steps 3 and 4 in the query phase of the algorithm used for locating the approximate 1-center perform the \textit{Group Enclosing Query} discussed in section 4.1. We use the approximate algorithm proposed by Li et al. [79]. It has a \( O(\log n + m) \) query cost in any fixed dimension, where \( n \) and \( m \) are the sizes of the point set \( P \) and the query set \( Q \) respectively. It computes the minimum enclosing ball (\( \text{MEB} \)) of the query points and then finds the nearest neighbor of the center of the ball from the set \( P \). We state an important theorem regarding the algorithm. For proofs, please refer to [79].

\[ p = \text{nn}(c, V_{de}) \]

Figure 4.10: A \textit{GEQ} example. The solid points are in \( Q_{de} \) and the hollow ones are in \( V_{de} \). Center \( c \) of \( \text{MEB}(c, Q_{de}) \) is snapped to its nearest neighbor (denoted by \( \text{nn}(c, V_{de}) \)) in \( V_{de} \).

**Theorem 4.7.1** Given a data structure that can do \( \sigma \)-approximate nearest neighbor queries, and another algorithm that can compute \( \rho \)-approximate minimum enclosing balls, the \textit{GEQ} algorithm computes \( \sqrt{\sigma^2 + \rho^2} \)-approximate solution.

We compute the exact \( \text{MEB} \) using CGAL (http://www.cgal.org) and \((1+\epsilon)\)-approximate \textit{NN} of the center of the \( \text{MEB} \) using STANN [34].

### 4.8 Experimental Setup

Our algorithm was implemented in C++ and MATLAB. The implementation of the exact algorithm and the graph partitioning part of our algorithm was written in C++
and compiled with g++ 4.1.2 with -O3 optimization. For constructing the graphs from the road networks, the BOOST (http://www.boost.org) library was used. The CGAL and STANN libraries were used for implementing the KH algorithm, computing the Minimum Enclosing Ball and the NN-search in the Euclidean space. FastMDS was implemented using MATLAB. The machine used has 8 Quad-Core AMD Opteron(tm) Processor 8378 with hyperthreading enabled. Each core has a L1 cache size of 512 KB, L2 of 2MB and L3 of 6MB with 128 GB total memory. The operating system was CentOS 5.3. All data was generated and stored as 64 bit C++ doubles. Currently, we use 24 cores of this machine for preprocessing and one core for the queries. In the next section we present two sets of graphs. In the first set, we show if the embedding that we have generated from the graphs is meaningful. To this end, we present our results from the embedding of the road networks from Tallahassee (1597 nodes and 2425 edges), Orlando (2954 nodes and 4602 edges), Tampa (2932 nodes and 4554 edges), Miami (6554 nodes and 10063 edges), Florida (1001982 nodes and 1327731 edges) and the entire United States (26136771 nodes and 32982669 edges). The number of nodes and edges given are obtained after removal of degree 2 nodes. In the second set, we show the error in approximation of the 1-center radius obtained from our approximate answer when compared against the exact answer.

4.9 Experimental Results

Once we embedded the shortest path metric into $\mathbb{R}^6$, we wanted to make sure that the embedding was meaningful. In these experiments, we use our $AbsR$ measure, because for the large data sets we do not have the information for all pairwise distances to compute the scaling factor. Although for the small sets we do not have this problem, we use the same measure for both. We sampled 10,000 pairwise distances from the road networks of the maps mentioned in the previous section. From the map of USA, we sampled 1000 pairwise distances, since the computation of 10,000 shortest paths was too expensive. Figure 4.11 show the $AbsR$ measures for the pairwise distances. Note that most of these pairs have very small $AbsR$ measures, although Miami, Florida and USA have higher $AbsR$ measures. This is due to the fact that they fall under our category of “large data”, hence FastMDS had to be used for their embedding, thus increasing the error factor.

Encouraged by these results, we next compared the exact radius computed using KH algorithm and the radius computed by our algorithm. This is shown in Figure 4.12. We use a slight abuse of notation in here. There are two comparisons done in these graphs. One is with the “absolute answer” computed by the KH algorithm, shown by “Absolute Distortion” in the graph. The second is with the exact “vertex answer”, shown by “Vertex Distortion”. We use the term “Mean Radii Distortion” for the measuring the error in approximation of our answer with the true radius. Let $R_{ab}$ and $R_v$ denote the true absolute and vertex radii and $R_{approx}$ denote the approximate radius. All the radii are shortest path ones, i.e. the shortest path distance between the
Table 4.1: Time chart in seconds for computing the 1-center for query sets of sizes $2^x, x = 1, \ldots, 10$

<table>
<thead>
<tr>
<th></th>
<th>$x = 1$</th>
<th>$x = 2$</th>
<th>$x = 3$</th>
<th>$x = 4$</th>
<th>$x = 5$</th>
<th>$x = 6$</th>
<th>$x = 7$</th>
<th>$x = 8$</th>
<th>$x = 9$</th>
<th>$x = 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tallahassee</td>
<td>Vertex</td>
<td>0.002</td>
<td>0.003</td>
<td>0.005</td>
<td>0.01</td>
<td>0.021</td>
<td>0.04</td>
<td>0.08</td>
<td>0.162</td>
<td>0.336</td>
</tr>
<tr>
<td></td>
<td>Absolute</td>
<td>0.006</td>
<td>0.013</td>
<td>0.031</td>
<td>0.053</td>
<td>0.132</td>
<td>0.306</td>
<td>0.809</td>
<td>2.052</td>
<td>4.874</td>
</tr>
<tr>
<td></td>
<td>Approximate</td>
<td>$10^{-7}$</td>
<td>$10^{-7}$</td>
<td>$10^{-6}$</td>
<td>$10^{-6}$</td>
<td>$10^{-6}$</td>
<td>$10^{-5}$</td>
<td>$10^{-5}$</td>
<td>$10^{-4}$</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Tampa</td>
<td>Vertex</td>
<td>0.002</td>
<td>0.005</td>
<td>0.010</td>
<td>0.021</td>
<td>0.043</td>
<td>0.087</td>
<td>0.175</td>
<td>0.358</td>
<td>0.721</td>
</tr>
<tr>
<td></td>
<td>Absolute</td>
<td>0.009</td>
<td>0.0268</td>
<td>0.06</td>
<td>0.134</td>
<td>0.239</td>
<td>0.505</td>
<td>1.252</td>
<td>2.545</td>
<td>5.449</td>
</tr>
<tr>
<td></td>
<td>Approximate</td>
<td>$10^{-6}$</td>
<td>$10^{-6}$</td>
<td>$10^{-6}$</td>
<td>$10^{-6}$</td>
<td>$10^{-6}$</td>
<td>$10^{-5}$</td>
<td>$10^{-5}$</td>
<td>$10^{-4}$</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Orlando</td>
<td>Vertex</td>
<td>0.003</td>
<td>0.005</td>
<td>0.010</td>
<td>0.021</td>
<td>0.043</td>
<td>0.087</td>
<td>0.175</td>
<td>0.357</td>
<td>0.732</td>
</tr>
<tr>
<td></td>
<td>Absolute</td>
<td>0.009</td>
<td>0.027</td>
<td>0.063</td>
<td>0.133</td>
<td>0.27</td>
<td>0.538</td>
<td>1.125</td>
<td>2.964</td>
<td>7.113</td>
</tr>
<tr>
<td></td>
<td>Approximate</td>
<td>$10^{-6}$</td>
<td>$10^{-6}$</td>
<td>$10^{-6}$</td>
<td>$10^{-6}$</td>
<td>$10^{-6}$</td>
<td>$10^{-5}$</td>
<td>$10^{-5}$</td>
<td>$10^{-4}$</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Miami</td>
<td>Vertex</td>
<td>0.007</td>
<td>0.014</td>
<td>0.028</td>
<td>0.056</td>
<td>0.112</td>
<td>0.226</td>
<td>0.459</td>
<td>0.923</td>
<td>1.864</td>
</tr>
<tr>
<td></td>
<td>Absolute</td>
<td>0.021</td>
<td>0.060</td>
<td>0.116</td>
<td>0.259</td>
<td>0.467</td>
<td>1.095</td>
<td>2.162</td>
<td>4.513</td>
<td>13.072</td>
</tr>
<tr>
<td></td>
<td>Approximate</td>
<td>$10^{-4}$</td>
<td>$10^{-4}$</td>
<td>$10^{-3}$</td>
<td>$10^{-3}$</td>
<td>$10^{-3}$</td>
<td>$10^{-3}$</td>
<td>$10^{-3}$</td>
<td>$10^{-3}$</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Florida</td>
<td>Vertex</td>
<td>4.88</td>
<td>9.855</td>
<td>19.95</td>
<td>39.92</td>
<td>79.44</td>
<td>159.94</td>
<td>321.94</td>
<td>656.38</td>
<td>1309.9</td>
</tr>
<tr>
<td></td>
<td>Absolute</td>
<td>7.190</td>
<td>17.007</td>
<td>35.276</td>
<td>68.288</td>
<td>138.7</td>
<td>235.54</td>
<td>434.16</td>
<td>818.44</td>
<td>1467.98</td>
</tr>
<tr>
<td></td>
<td>Approximate</td>
<td>0.023</td>
<td>0.024</td>
<td>0.022</td>
<td>0.023</td>
<td>0.027</td>
<td>0.024</td>
<td>0.025</td>
<td>0.037</td>
<td>0.048</td>
</tr>
</tbody>
</table>

1-center and the furthest point from it. Then the figures in the y-axes of the graphs is the average from 50 runs of $R_{approx}/R_{ab}$ for computing the Absolute Distortion and $R_{approx}/R_v$ for computing the Vertex Distortion. We see that we are more closer to the vertex answer than the absolute answer. This is because $R_{approx}$ is greater than both $R_{ab}$ and $R_v$, and $R_v \geq R_{ab}$. Hence $R_{approx}/R_v \leq R_{approx}/R_{ab}$. The increase in distortion for both the answers occur when the queries are fed from the points having large distortions when embedded. The road networks of Orlando and Tampa are denser than that of Tallahassee. Thus the vertex answer is close to an absolute answer in the former cases. That also explains why the difference between the vertex answers and absolute answers are closer in cases of Orlando and Tampa unlike Tallahassee. The other thing worthy of note is that for Miami and Florida, the distortions are higher than those of Tallahassee, Tampa or Orlando. We always are above the 1.1 mark in case of Miami unlike the other three. This is because Miami and Florida have more points that are badly embedded due to the fact that they were embedded piecewise using FastMDS. The timings for both algorithms are shown in Table 4.1. As stated before, a cubic time algorithm is no match to our algorithm.

4.9.1 Interface

In order to provide easy access to generic one-center queries, a web interface was developed. Figure 4.13 shows a snapshot of our demo website (http://maya.cs.fsu.edu). The demonstration uses a subset of the map of Tallahassee limited by the box on the map. For selecting query points, a nearest neighbor lookup is performed between the clicked point and the nodes of the network in coordinate space. Note that Steiner points are included at this stage, to better match query points on an edge of the
Figure 4.11: AbsR values for 10,000 pairwise distances sampled randomly from Tallahassee, Orlando, Tampa, Florida and Miami. AbsR values are shown for 1000 pairwise distances from continental USA.
Figure 4.12: Error in approximation of the 1-center answer computed by our algorithm.
network. This answer is displayed to the user on the map. After the query points are obtained the one-center is calculated as described in this chapter. Answers are reported in terms of latitude and longitude and is then plotted on the map. For comparison purposes, the exact answer (the small size of the data set allows us to do so) can also be calculated and displayed. However, this ability is only available in the demo website. We have another website for querying one-center solutions approximately, but instantly on the road network of Florida (http://maya.cs.fsu.edu/map). Very soon, we will provide an interface for queries on the road network of the entire continental USA.

![Figure 4.13: Screenshot of the Interface. The approximate one center to the query points A, B, C, and D is shown.](image)

4.10 Conclusion

This work presented the first practical solution to the one-center problem. Our experiments show that the error in approximation with the true one-center is only by a constant factor. One of the improvements that can be made on this work is to improve the approximate answer by using shortest path oracles.
CHAPTER 5

MINIMUM ERROR RATE TRAINING
BY SAMPLING THE TRANSLATION LATTICE

In this chapter, we propose a method \cite{26} alternative to classical MERT, consisting in sampling a list of candidate translations from the probability distribution induced by the translation lattice. This simple method produces a list of candidates more representative of the complete distribution than an N-best list, side-stepping the intricacies of propagating envelopes throughout the lattice. Computational complexity increases only marginally over the N-best list approach, while still yielding significant improvements in final translation quality.

5.1 Related Work

Since its introduction, \cite{99} there has been various suggestions for optimizing the MERT criterion. Zens et al. \cite{142} use the MERT criterion to optimize the N-best lists using the Downhill Simplex Algorithm \cite{107}. But the Downhill Simplex Algorithm loses its robustness as the dimension goes up by more than $10^{\text{\cite{85}}}$. Deterministic Annealing was suggested by Smith and Eisner \cite{119} where the authors propose to minimize the expected loss or risk. They define the expectation using a probability distribution over hypotheses that they gradually anneal to focus on the 1-best hypothesis. Different search strategies were investigated by Cer et al. \cite{21}. Work has been done to investigate a perceptron-like online margin training for statistical machine translation \cite{131}. Building on this chapter, the most recent work to our knowledge has been done by Chiang et al. \cite{30}. They explore the use of the Margin Infused Relaxed Algorithm (MIRA) \cite{38, 37} algorithm instead of MERT. Macharey et al. \cite{85} propose a new variation of MERT where the algorithm is tuned to work on the whole phrase lattice instead of N-best list only. The new algorithm constructs the error surface of all translations that are encoded in the phrase lattice. They report significant convergence improvements and BLEU score gains over N-best MERT when trained on NIST 2008 translation tasks. More recently, this algorithm was extended to work
with hypergraphs encoding a huge number of translations produced by MT systems based on Synchronous Context Free Grammars [74]. All the methods cited here work on either N-best lists or from whole translation lattices built by the decoder. To our knowledge, none of them proposes sampling translations from the lattice.

5.2 Sampling Candidate Translations from the Lattice

In this section we first start by providing an intuition of why we believe it is a good idea to sample from the translation lattice, and then describe in detail how we do it.

5.2.1 An intuitive explanation

The limited scope of n-best lists rules out many alternative translations that would receive the highest score for some values of the parameter vector. The complete set of translations that can be produced using a fixed phrase table (also called reachable translations) for a given source sentence can be represented as a set of vectors in the space spanned by the feature functions (Fig. 5.1). Not all such translations stand a chance to receive the highest score for any value of the parameter vector, though. Indeed, if translations $h$, $h'$ and $h''$ are such that $h_k \leq h'_k \leq h''_k$ for all feature $k$, then there is no value of $\bar{\lambda}$ that will give to $h'$ a score higher than both $h$ and $h''$. The candidates that would rank first for some value of the $\bar{\lambda}$ parameter vector are those on the convex envelope of the overall candidate set. We know of no effective way to generate this convex envelope in polynomial time. The set of candidates represented by the decoder lattice is a subset (enclosed in the larger dashed polygon in the figure) of this set. This subset is biased to contain translations ranking high according to the values of the parameter vector (the direction labelled with $\lambda$) used to produce it, because of the pruning strategies that guide the construction of the translation lattice. Both the N-best list and our proposed random sample are further subsets of the set of translations encoded in the lattice. The N-best list is very biased towards translations that score high with the current choice of parameters: its convex envelope (the smaller dashed polygon) is very different from the one of the complete set of translations, and also from that of the translations in the lattice. The convex envelope of a random sample from the translation lattice (the dotted polygon in the figure), will generally be somewhat closer to the envelope of the whole lattice itself.

The curves in the figure indicate regions of constant loss (e.g. iso-BLEU score, much more irregularly shaped in reality than in the drawing). For this sentence, then, the optimal choice of the parameters would be around $\lambda^*$. Performing an optimization step based on the random sample envelope would result in a more marked update ($\lambda'_{\text{sample}}$) in the direction of the best parameter vector than if an N-best list is used ($\lambda'_{\text{N-best}}$).
Figure 5.1: Envelope of the set of reachable translations where the model has two feature functions $h_1$ and $h_2$. The envelope of the lattice is the outer dashed polygon, while the envelope of the N-best list is the inner one. Using the whole lattice as translation pool will result in a more marked update towards the optimal parameters. The random sample from the lattice is enclosed by the dotted line. If we use it, we can intuitively expect updates towards the optimum of intermediate effectiveness between those of the N-best list method and those of the lattice method.

Notice that Figure 5.1 portraits a situation with only two features, for obvious reasons. In practice the number of features will be substantially larger, with values between five and twenty being common practice. In real cases, then, a substantially larger fraction of reachable translations will tend to lie on the convex envelope of the set, and not inside the convex hull [41].

5.2.2 The sampling procedure

We propose to modify the standard MERT algorithm and sample N candidates from the translation lattice according to the probability distribution over paths induced by the model, given the current setting of the $\lambda$ parameters, instead of using an N-best list. The sampling proceeds from the root node of the lattice, corresponding to an empty translation candidate covering no words of the source, by choosing step by step the next edge to follow. The probability distribution for each possible follow-up is the posterior probability of following the edge given the path prefix derived from the lattice: it is obtained via a preliminary backward sweep.

Since feature functions are incremental over the edges by design, the non-normalized probability of a path is given by:

$$P(e_1, \ldots, e_m) = e^{\sum_{i=1}^{m} \sigma(e_i)}$$

where
\[ \sigma(e_i) = \sum_{k=1}^{K} \lambda_k h_k(e_i) \] (5.2)

is the score of edge \( e_i \). With a small abuse of notation we will also denote it as \( \sigma(n_{j,k}) \), where it is intended that \( e_i \) goes from node \( n_j \) to node \( n_k \). Let’s denote with \( \sigma(n_i) \) the score of node \( n_i \), i.e. the logarithm of the cumulative unnormalized probability of all the paths in the lattice that go from node \( n_i \) to a final node. The unnormalized probability of selecting node \( n_j \) starting from \( n_i \) can then be expressed recursively as follows:

\[ S(n_j|n_i) \approx e^{(\sigma(n_j)+\sigma(n_{i,j}))} \] (5.3)

The scores required to compute this sampling probabilities can be obtained by a simple backward pass in the lattice. Let \( P_i \) be the set of successors of \( n_i \). So the total unnormalized log-probability of reaching a final state (i.e. with a complete translation) from \( n_i \) is given by the equation below.

\[ \sigma(n_i) = \log\left( \sum_{n_j \in P_i} e^{(\sigma(n_j)+\sigma(n_{i,j}))} \right) \] (5.4)

where we set \( \sigma(n_i) = 0 \) if \( P_i = \emptyset \), that is if \( n_i \) is a final node. At the end of the backward sweep, \( \sigma(n_0) \) contains the unnormalized cumulative probability of all paths, i.e. the partition function. Notice that this normalising constant cancels out when computing local sampling probabilities for traversed nodes in the lattice.

Once we know the transition probability (Eq. 5.3) for each node, we sample by starting in the root node of the lattice and at each step randomly selecting among its successors, until we end in the final node. The whole sampling procedure is repeated as many times as the number of samples sought. After collecting samples for each sentence, the whole list is used to grow the translation pool.

Notice that when using this sampling method it is no longer possible to use the stability of the translation pool as a stopping criterion. The MERT algorithm must thus be run either for a fixed number of iterations, or until the norm of the update to the parameter vector goes below a threshold.

### 5.2.3 Time Complexity Analysis

For each line search in the inner loop of the MERT algorithm, all methods considered here need to compute the projection of the convex envelope that can be scanned by leaving all components unchanged but one\(^1\). If we use either N-best lists or random samples to form the translation pool, and \( M \) is the size of the translation pool, then computing the envelope can be done in time \( O(M \log M) \) using the SweepLine algorithm reproduced as Algorithm 1 in [85]. As shown in the same article, the lattice

\(^1\)In general, moving along a 1-dimensional subspace of the parameter space.
method for computing the envelope is \( O(|V||E| \log |E|) \), where \( V \) is the vertex set of the lattice, and \( E \) is its edge set. In standard decoders there is a maximum limit \( D \) to the allowed distortion, and lattice vertices are organized in \( J \) priority queues \(^2\) of size at most \( a \), where \( J \) is the length of the source sentence and \( a \) is a parameter of the decoder set by the user. Also, there is a limit \( K \) to the maximum number of source words spanned by a phrase, and only up to \( c \) alternative translations for a same source phrase are kept in the phrase table. Under these standard conditions, the number of outgoing edges \( E' \) from each lattice vertex can be bounded by a constant. A way to see this is by considering that if an hypothesis is extended with a phrase, then the extended hypothesis must end up in a stack at most \( K \) stacks to the right of the original one. There are only \( aK \) places in these stacks, so it must be \( |E'| \leq aK \). Since the number of edges leaving each node is bounded by a constant, it is \( |E| = \Theta(|V|) \), and the lattice method is \( O(|V|^2 \log(|V|)) \). The maximum number of vertices in the lattice is limited by the capacity of the stacks: \( |V| \leq aJ \). This eventually leads to a complexity of \( O(J^2 \log J) \) for the inner loop of the lattice method.

It is interesting to observe that the complexity is driven by the length of the source sentence in the case of the lattice method, and by the size of the translation pool in the case of both the N-best list method and the random sampling method. The latter two methods are asymptotically more effective as long as the size of the sample/N-best list grows sub-quadratically in the length of the sentence. In most of our experiments we keep the size of the sample constant, independent of the length of the sentence, but other choices can be considered. Since the number of reachable translations grows with the length of the source sentence, length-independent samples explore a smaller fraction of the reachable space. Generating samples (or n-best lists) of size increasing with the length of the source sentence could thus lead to more homogeneous sampling, and possibly a better use of CPU time.

We have so far compared methods in term of the complexity of the innermost loop: the search for a global optimum along a line in the parameter space. This is indeed the most important analysis, since the line search is repeated many times. In order to complete the analysis, we also compare the different methods in terms of the operations that need be performed as part of the outer iteration, that is upon redecoding the development set with a new parameter vector.

The N-best list method requires simply constructing an N-best list from the lattice. This can be done in time linear in the size \( J \) of the sentence and in \( N \) with a backward sweep in the lattice.

The sampling method requires sampling \( N \) times the lattice according to the probability distribution induced by the weights on its edges. We use a dynamic programming approach for computing the posterior probabilities of traversing edges. In this phase we visit each edge of the lattice exactly once, hence this phase is linear in the number of edges in the lattice, hence under the standard assumptions above in the length \( J \) of the sentence. Once posterior probabilities are computed for the

\(^2\)Traditionally referred to as stacks.
lattice, we need to sample $N$ paths from it, each of which is composed of at most $J$ edges\footnote{We assume all phrase pairs cover at least one source word.}. Under standard assumptions, randomly selecting the next edge to follow at each lattice node can be done in constant time, so the whole sampling is also $O(NJ)$, like extracting the N-best list.

No operation at all is required by the lattice method in the outer loop, since the whole lattice is passed over for envelope propagation to the inner loop.

### 5.3 Experimental Results

Experiments were conducted on the Europarl corpus with the split used for the WMT-08 shared task (Europarl training and test condition) for the language pairs English-French (En-Fr), English-Spanish (En-Es) and English-German (En-De), each in both directions. Training corpora contain between 1.2 and 1.3 million sentence pairs each, development and test datasets are of size 2,000. Detailed token and type statistics can be found in [19]. The Moses decoder [69] was used for generating lattices and n-best lists. The maximum number of decoding iterations was set to twelve. Since Moses was run with its lexicalised distortion model, there were 14 features. Moses L1-normalises the parameter vector: parameter scaling only marginally affects n-best list construction (via threshold pruning during decoding), while it substantially impacts sampling.

For each of the six configurations, we compared the BLEU score on the test data when optimizing feature weights with MERT using n-best and random samples of size 100 and 200. In all cases we used 20 random restarts for MERT. Results are presented in Table 5.1. We also ran non systematic experiments on some of the configurations with larger samples and n-best lists, with results changing very little from the respective 200 cases: we do not report them here.

Learning curves (BLEU on the development set) are shown in Figure 5.2. Learning curves for the other tested language pairs follow a similar pattern.

### 5.4 Analysis of results

All differences of the test scores between optimizing the parameters using nbest-200 lists and from randomly sampled lists of size 200 were found to be statistically significant at 0.05 level at least. We used Approximate Randomization Test [110] for the purpose, random sampling being done 1000 times.

Somewhat surprisingly, while random sampling with sample size of 200 yields overall the best results, random sampling with size 100 give systematically worse results than n-best lists of the same size. We conjectured that n-best lists and random samples could have complementary advantages. Indeed, it seems intuitive that a good translation pool should be sufficiently varied, as argued in Section 5.2.1. However it
Figure 5.2: Learning curves (BLEU on the development set) for different tested conditions for English to French (top) and French to English (bottom).
should also stand high chances to contain the best reachable translation, or translations close to the best. It might thus be that 100-best lists are unable to provide diversity, and random samples of size 100 to guarantee sufficient quality.

In order to test this conjecture we repeated our experiments, but at each iteration we used the union of a 100 random sample and a 100 n-best list. Results for this experiments are in Table 5.2. The corresponding results with random samples of size 200 are also repeated to ease comparison. Depending on the language pair, improvements over random sampling range from 0.17 (En-Es) to 0.44 (Fr-En) BLEU points. Improvements over 200-best lists range from 0.68 (De-En) to 0.89 (Fr-En) BLEU points. These results indicate quite clearly that N-best lists and random samples contribute complementary information to the translation pool: indeed, in most cases there is very little or no overlap between the two.

Table 5.2: Test set BLEU Scores for the same “Source-Target” pairs using a mixed strategy combining a 100 N-best list and a random sample of size 100 after each round of decoding.

<table>
<thead>
<tr>
<th>Source-Target</th>
<th>Mixed 100 + 100</th>
<th>RS-200</th>
</tr>
</thead>
<tbody>
<tr>
<td>En-Fr</td>
<td>33.17</td>
<td>32.76</td>
</tr>
<tr>
<td>Fr-En</td>
<td>33.35</td>
<td>32.91</td>
</tr>
<tr>
<td>En-Es</td>
<td>30.37</td>
<td>30.19</td>
</tr>
<tr>
<td>Es-En</td>
<td>32.04</td>
<td>31.66</td>
</tr>
<tr>
<td>En-De</td>
<td>21.31</td>
<td>20.93</td>
</tr>
<tr>
<td>De-En</td>
<td>27.98</td>
<td>27.62</td>
</tr>
</tbody>
</table>

Convergence curves show that RS-200, NB-100 and M-200 (i.e. the hybrid combination) systematically converge to higher BLEU scores, on the development set and on their respective translation pools, than RS-100 and NB-200. Notice however that it is misleading to compare scores across different translation pools, especially if these have substantially different sizes. On the one hand adding more candidates increases the chances of adding one with high contribution to the corpus BLEU, and can thus
increase the achievable value of the objective function. On the other hand, adding more candidates reduces the freedom MERT has to find parameter values selecting high-BLEU candidates for all sentences. To see this, consider the extreme case when the translation pools are all of size one and are provided by an oracle that gives the highest-BLEU reachable translation for each sentence: the objective surface is uninformatively flat, all values of the parameters are equally good, and the BLEU score on the devset is the highest achievable one. If now we add to each translation pool the second-best BLEU-scoring candidate, BLEU will be maximized in a half-space for each sentence in the development set: MERT will try to select $\lambda$ in the intersection of all the half-spaces, if this is not empty, but will have to settle for a lower-scoring compromise otherwise. The larger the translation pools, the more difficult it becomes for MERT to “make all sentences happy”. A special case of this is when adding more candidates extends the convex envelopes in such a way that the best candidates fall in the interior of the convex hull. It is difficult to tell which of the two opposing effects (the one that tends to increase the value of the objective function or the one that tends to depress it) is stronger in any given case, but from the convergence curves it would seem that the first prevails in the case of random samples, whereas the second wins in the case of n-best lists. In the case of random samples going from size 100 to 200 systematically leads to higher BLEU score on the devsets, as more high-BLEU candidates are drawn. In the case of n-best lists, conversely, this leads to lower BLEU scores, as lower-BLEU (in average) candidates are added to translation pools providing a sharper representation of the BLEU surface and growing MERT out of the “delusion” that a given high BLEU score is actually achievable.

In the light of this discussion, it is interesting to observe that the value achieved by the objective function on the development set is only a weak predictor of performance on the test set, e.g. M-200 never converges to values above those of NB-100, but is systematically superior on the test data.

In [85] the authors observe a dip in the value of the objective function at the first iteration when training using n-best lists. We did not observe this behaviour in our experiments. A possible explanation for this resides in the larger size of the n-best lists we use (100 or 200, compared to 50 in the cited work) and in the smaller number of dimensions (14 instead of 20-30).

We hinted in Section 5.2.3 that it would seem reasonable to use samples/nbest-list of size increasing with the length of the source sentence, so as to sample reachable translations with a more uniform density across development sentences. We tested this idea on the French to English condition, making samples size depend linearly on the length of the sentence, and in such a way that the average sample size is either 100 or 200. For average sample size 100 we obtained a BLEU of 31.55 (compared to 31.77 with the constant-size 100 random sample) and for average size 200 31.84 (32.46 in the corresponding constant-size condition). While partial, these results are not particularly encouraging w.r.t. using variable size samples.

Finally, in order to assess the stability of the proposed training procedure across variations in development datasets, we experimented with extracting five distinct
devsets of size 2,000 each for the French to English RS-200 condition, keeping the test set fixed: the maximum difference we observed was 0.33 BLEU points.

\section{5.5 Conclusion}

We introduced a novel variant to the well-known MERT method for performing parameter estimation in Statistical Machine Translation systems based on log-linear models. This method, of straightforward implementation, is based on sampling candidates from the posterior distribution as approximated by an existing translation lattice in order to progressively expand the translation pool that shapes the optimization surface. This method compares favorably against existing methods on different accounts. Compared to the standard method by which N-best lists are used to grow the translation pool, it yields empirically better results as shown in our experiments, without significant penalties in terms of computational complexity. These results are in agreement with the intuition that the sampling method introduces more variety in the translation pool, and thus allows to perform more effective parameter updates towards the optimum. A hybrid strategy, consisting in combining N-best lists and random samples, brings about further significant improvements, indicating that both quality and variety are desirable in the translation pool that defines the optimization surface. A possible direction to investigate in the future consists in generalizing this hybrid strategy and combining random samples where the probability distribution induced on the lattice by the current parameters is scaled by a further temperature parameter $\beta$:

\begin{equation}
\frac{P'(e, a|f)}{P'(e, a|f)} \propto P(e, a|f)^\beta
\end{equation}

where for $\beta = 1$ the random samples used in this chapter are obtained, for $\beta$ tending to infinite the distribution becomes peaked around the single best path, thus producing samples similar to N-best lists, and samples from other real values of the temperature can be combined.

Compared to the method using the whole lattice, the proposed approaches have a substantially lower computational complexity under very broad and common assumptions, and yet yield translation quality improvements of comparable magnitude over the baseline N-best list method.
CHAPTER 6

CONCLUSIONS AND CONTRIBUTIONS

This work attempted to expand the tool set for solving common applications of nearest neighbor search problems and a classical statistical machine translation problem. By suitable engineering of existing algorithms, the work showed how to obtain better practical running times without compromising the theoretical guarantees.

The work on the GMST problem made some important observations. We conjectured that the particular well separated pair decomposition algorithm used is not relevant to the correctness of our algorithm, thus the use of a tree that offers better clustering might make the algorithm more efficient. We showed that spatial decomposition using a fair split is more favorable than using a quadtree decomposition, since the number of clusters that the algorithm has to deal with is fewer. This is because the clustering produced by the FST was better, and the WSPD algorithm could find “well-separated” pairs without going as deep into the tree. We tried with different $L_p$ values, however, our experiments using a value of $p$ other than 2 did not show any change in the relative behavior of the algorithms, so we did not report the results for metrics other than $L_2$. Our experiments using the k-nearest neighbor approach turned out to be slower than both the algorithms discussed here for low dimensions. Even in higher dimensions, although the number of well-separated pairs kept being a bottleneck in the performance, the k-nearest neighbor approach turned out to be even worse. One open problem that arises from this work is whether the well separated decomposition can be built in a way such that a partial GMST can be determined even before one starts looking at the pairwise cluster distances. This, we believe, will improve the running time by a considerable amount. The practical impact of other parallel sort and partition algorithms can be incorporated to scale the algorithm better.

The 1-Center project opens up many new and interesting avenues of research. One immediate open problem is how to reduce the error in approximation of the 1-center answers for large data sets. One can find the lowest common ancestor in the partition tree and use the embedding of that partition to locate the answer. As for the embedding, we are looking into other methods, such as proposed by Zhang.
et al. [143], as well as ways of embedding the network into $L_1$ metric. Our initial experiments on small graphs gave encouraging results with the $L_1$ metric. As for the semidefinite programming approach, it is to be seen whether we can use semidefinite relaxation [70] techniques to handle large data sets. The applications of 1-center can be extended into $k$-center problems. For example, instead of building one touristic office, the problem can be to build $k$ such offices, such that the office is the closest one to the hotels that are within the ball defined by each of the centers. The idea can be extended to find ways of solving the $k$-center/$k$-means problems for points in motion on road networks. We already have done preliminary experiments with k-means on our embeddings and seen encouraging results. We hope that someday this method is used to save fuel and reduce pollution in the real world.

The MERT method presented in this thesis operates on the translation lattices generated by Phrase-Based SMT decoders. The extension to translation forests generated by hierarchical decoders [29] seems straightforward. In that case, the backward sweep for propagating unnormalized posterior probabilities is replaced by a bottom-up sweep, and the sampling now concerns (binary) trees instead of paths, but the rest of the procedure is substantially unchanged. We conjecture however that the extension to translation forests would be less competitive compared to working with the whole packed forest (as in [74]) than lattice sampling is compared to working with the whole lattice. The reason we believe this is that hierarchical models lead to much more spurious ambiguity than phrase-based models, so that both the N-best method and the sampling method explore a smaller portion of the candidate space compared to the compact representation of all the candidate translations in a beam.

The fields of computational geometry and statistical machine translation offer a wide array of problems which are applicable to a multitude of other fields of science. By improving on solutions to these problems, hopefully both the fields of computational geometry and SMT, as well as the fields that use it, can be advanced. The goal of this work was to be just such an improvement, and it is the hope of the author that other researchers in these fields will find the results presented here useful.
APPENDIX A

NATIONAL SCIENCE FOUNDATION
INNOVATION CORPS AWARD

The NSF Innovation Corps (I-Corps) is a set of activities and programs that prepare scientists and engineers to extend their focus beyond the laboratory and broadens the impact of select, NSF-funded, basic-research projects. The main aim of this program is to train academicians to bring their work to the market for commercialization. A team of three comprised of the Principal Investigator (PI), Entrepreneurial Lead (EL) and a Mentor can participate in the program. We believe that the work presented in Chapter 4 has business potential. We submitted a proposal in response to the NSF I-Corps program and are proud to have received the award of $50,000. Our team comprises of Dr. Piyush Kumar as the PI, Mr. Tyler Tatum from RippleManagement Inc. (www.ripplemgmt.com) as the Mentor and Mr. Samidh Chatterjee as the EL. Below we describe in brief why we believe the work can be commercialized.

A.1 Potential Commercial Impact

A.1.1 Customer Profile

We envision the common people as our main customers. Specifically, groups of 5 or more people who want to meet at a central location will be our focus. There will be two versions of our software, namely a web-based solution and a mobile app. We do not plan to charge the common users for our web based application, rather charge businesses who might want to advertise on the webpage. Another set of customers will be professional users who are responsible for planning meetings as part of their jobs. For the mobile application, the common users will have the option to be charged to download the software or view advertisements. The end result of the project will also help reduce environmental pollution by controlling fuel emission.
A.1.2 Customer Need

The 1-Center project aims to improve convenience, minimize fuel consumption and travel times by finding common meeting locations for groups of users. Formally, it aims to solve the 1-center (sum) problem, where users are looking to minimize the maximum (total) distance/time traveled. If there’s a registered business close to the calculated location, we can suggest that business as the meeting point and display its information. Placing k facilities in a region such that each person can be assigned to one of the sites is an instance of k-center/sum problem. Carbon offsets are voluntary payments used to reduce global Greenhouse Gas total instead of making radical reductions to own lifestyle. Supporting our service helps reduce carbon footprint. There are several established companies selling ‘credits’ to offset your carbon footprint (www.nativeenergy.com).

A.1.3 How the needs are currently met?

The current state-of-the-art is limited in terms of the time it takes to optimize many different points for the minimum travel time and fuel consumption. Current solutions either simplify the problem by doing an “as the crow flies” analysis or require several hours or days to create a solution. Companies providing such services now either are limited to calculating the midpoint of two or three points (e.g. www.meetways.com & www.mezzoman.com/) or they provide very crude midpoint solutions (e.g. http://geomidpoint.com or http://a.placebetween.us ).

A.1.4 Payment Amount from Customers

First, we will employ traditional online advertising. Second, we will offer premium content to those who want significantly more functionality than the casual user. These users could be professional meeting planners who are interested in storing information on the site or having our system link to corporate systems they utilize. Third, we will look to license our capabilities to other internal or external websites who could leverage our solution to expand on their offerings. Large corporations, for example, may want to leverage our solution on their internal site in order to make meeting planning more efficient among coworkers. Larger sites like Google or Facebook may consider this as an addition to their current offerings as well.

A.2 Deliverables

The award is for 6 months, and at the end of the 6-month period we plan to produce the following deliverables. An enhanced web based application for road network graphs, scaling to as large as the state of Florida, as well as the whole continental United States will be delivered. We will be presenting the rationale behind our method of embedding. There are various optimization functions one can use in
order to do the embedding. Each of these optimization functions is a measure of the error that is incurred when the shortest path distance between two vertices in the graph is approximated by the Euclidean distance between their corresponding representations in the embedded space. Our presentation will rationalize the choice of the optimization function, as well as the dimension of the embedded space. The tools used in the query phase will also be explained. The project kicked off from October 1, 2012 and is supposed to be completed by March 31, 2013.
REFERENCES


BIOGRAPHICAL SKETCH

Samidh Chatterjee was born in Kolkata, India. After completing his undergraduate studies in India, he moved to the University of Windsor, Canada, where he earned his M.Sc. in Computer Science. He joined The Florida State University as a PhD student in 2007. He will be joining Airsage Inc., in Atlanta, GA as a Research Scientist following the completion of his PhD program.