PARALLEL AND NETWORK ALGORITHMS AND APPLICATIONS FOR STEINER TREES
AND VORONOI DIAGRAM

A dissertation submitted to
Kent State University in partial
fulfillment of the requirements for the
degree of Doctor of Philosophy

by

Rashid Bin Muhammad

December, 2009
Dissertation written by

Rashid Bin Muhammad

B.Sc. and M.Sc., University of Karachi, 1990
M.S., Murray State University, 1994
M.S., Kent State University, 2003
Ph.D., Kent State University, 2009

Approved by

Johnnie W. Baker, Chair, Doctoral Dissertation Committee
Feodor F. Dragan, Members, Doctoral Dissertation Committee
Paul F. Farrell
Mohammad K. Khan
Sameul N. Sprunt

Accepted by

Robert A. Walker, Chair, Department of Computer Science
John R. D. Stalvey, Dean, College of Arts and Sciences
# TABLE OF CONTENTS

LIST OF FIGURES ......................................................... vii

LIST OF TABLES .......................................................... viii

ACRONYMS ................................................................. x

ACKNOWLEDGEMENTS ...................................................... xi

1 INTRODUCTION ............................................................. 1
   1.1 About this Work .................................................... 1
   1.2 Motivation .......................................................... 1
   1.3 Notations and Background Information .............................. 5
   1.4 Structure of the Thesis and Main Contributions ................. 5

2 PRELIMINARIES ............................................................ 7
   2.1 Euclidean Steiner Problem ......................................... 7
      2.1.1 Relevant Steiner Tree Properties ............................. 7
      2.1.2 Relevant Steiner Tree Results ................................. 8
   2.2 Voronoi Diagram .................................................... 9
   2.3 Delaunay Triangulations .......................................... 10
   2.4 Gabriel Graph ...................................................... 12
   2.5 Relative Neighborhood Graph .................................. 12
   2.6 Geometric Graph .................................................. 13
   2.7 Range Assignment ................................................ 13
   2.8 Neighborhood ..................................................... 13
   2.9 Right Hand Rule ................................................ 14
   2.10 Communication Graph ............................................ 14
### 3 Computational and Mathematical Models

#### 3.1 Introduction

- **3.1.1 Chapter Guide**

#### 3.2 Server-Client Paradigm

#### 3.3 Long-Distance Model of Computation

#### 3.4 Hypercube Model of Computation

#### 3.5 Bibliographical Remarks

### 4 Parallel Steiner Tree

#### 4.1 Introduction

- **4.1.1 Our Contribution**

- **4.1.2 Chapter Guide**

#### 4.2 Overall Strategy

#### 4.3 Parallel Algorithm

- **4.3.1 Complexity Analysis**

- **4.3.2 Correctness**

#### 4.4 Execution Time Analysis

#### 4.5 Simulation

- **4.5.1 General Technique**

- **4.5.2 Halting Condition**

- **4.5.3 Simulation Variables**

- **4.5.4 Overall Strategy**

#### 4.6 Algorithm ServerQueue

#### 4.7 Illustrative Example

#### 4.8 Summary

### 5 Parallel Voronoi Diagram

#### 5.1 Introduction

- **5.1.1 Our Contribution**

- **5.1.2 Chapter Guide**
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The Steiner tree</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Full Steiner trees</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>A Torricelli point</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>The Quadrilateral abcd.</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>Two Steiner Points Addition</td>
<td>9</td>
</tr>
<tr>
<td>2.6</td>
<td>The Voronoi Diagram</td>
<td>10</td>
</tr>
<tr>
<td>2.7</td>
<td>Delaunay Triangulations and its Dual</td>
<td>11</td>
</tr>
<tr>
<td>2.8</td>
<td>Delaunay Triangles and Voronoi vertices</td>
<td>11</td>
</tr>
<tr>
<td>2.9</td>
<td>Gabriel Graph (GG)</td>
<td>12</td>
</tr>
<tr>
<td>2.10</td>
<td>Relative Neighborhood Graph (RNG)</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>Server-Client Computational Model.</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>Transmitting Range</td>
<td>18</td>
</tr>
<tr>
<td>3.3</td>
<td>Hypercube Computational Model.</td>
<td>19</td>
</tr>
<tr>
<td>6.1</td>
<td>Communication Graph</td>
<td>46</td>
</tr>
<tr>
<td>7.1</td>
<td>Nonplanar (LDel^1)</td>
<td>59</td>
</tr>
<tr>
<td>7.2</td>
<td>Li’s Solution</td>
<td>60</td>
</tr>
<tr>
<td>7.3</td>
<td>Discard all Edges</td>
<td>61</td>
</tr>
<tr>
<td>7.4</td>
<td>Our proposed Solution</td>
<td>62</td>
</tr>
</tbody>
</table>
LIST OF TABLES

4.1 Vectors Status. ................................................................. 36
4.2 Simulation Results. ......................................................... 36
List of Algorithms

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Overall Structure.</td>
<td>4</td>
</tr>
<tr>
<td>4.1</td>
<td>Algorithm Server (UPGRADE)</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>Algorithm Client (SEARCH)</td>
<td>24</td>
</tr>
<tr>
<td>4.3</td>
<td>Algorithm Server (Queue)</td>
<td>34</td>
</tr>
<tr>
<td>5.1</td>
<td>Construction of Voronoi Diagram on Hypercube</td>
<td>40</td>
</tr>
<tr>
<td>5.2</td>
<td>Construction of Dividing Polygonal Chain</td>
<td>40</td>
</tr>
<tr>
<td>6.1</td>
<td>ADD_NODE</td>
<td>49</td>
</tr>
<tr>
<td>6.2</td>
<td>Range Assignment Approximation</td>
<td>51</td>
</tr>
<tr>
<td>7.1</td>
<td>Connected Graph</td>
<td>61</td>
</tr>
<tr>
<td>7.2</td>
<td>Greedy Traversal</td>
<td>64</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>Convex Hull</td>
<td></td>
</tr>
<tr>
<td>DT</td>
<td>Delaunay Triangulation</td>
<td></td>
</tr>
<tr>
<td>ESMT</td>
<td>Euclidean Steiner Minimal Tree</td>
<td></td>
</tr>
<tr>
<td>ESP</td>
<td>Euclidean Steiner Problem</td>
<td></td>
</tr>
<tr>
<td>FST</td>
<td>Full Steiner Tree</td>
<td></td>
</tr>
<tr>
<td>GG</td>
<td>Gabriel Graph</td>
<td></td>
</tr>
<tr>
<td>GRASP</td>
<td>Greedy Randomized Adaptive Search Procedure</td>
<td></td>
</tr>
<tr>
<td>LDel($k$)</td>
<td>k-Localized Delaunay Graph</td>
<td></td>
</tr>
<tr>
<td>LDG</td>
<td>Local Delaunay Graph</td>
<td></td>
</tr>
<tr>
<td>MST</td>
<td>Minimum Spanning Tree</td>
<td></td>
</tr>
<tr>
<td>RA</td>
<td>Range Assignment</td>
<td></td>
</tr>
<tr>
<td>RDG</td>
<td>Restricted Delaunay Graph</td>
<td></td>
</tr>
<tr>
<td>RNG</td>
<td>Relative Neighborhood Graph</td>
<td></td>
</tr>
<tr>
<td>SMT</td>
<td>Minimum Steiner Tree</td>
<td></td>
</tr>
<tr>
<td>TSP</td>
<td>Traveling Sales Person Problem</td>
<td></td>
</tr>
<tr>
<td>UDG</td>
<td>Unit Disk Graph</td>
<td></td>
</tr>
<tr>
<td>VD</td>
<td>Voronoi Diagram</td>
<td></td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I would like to sincerely thank my advisor, Dr. Johnnie Baker, for his constant support and encouragement throughout my dissertation and for being my friend and mentor. I would like to thank committee members Dr. Paul Farrell and Dr. Feodor Dragan of Computer Science, Dr. Mohammad Kazim Khan and Dr. Richard Aron of Mathematical Sciences, and Dr. Samuel Sprunt of Physics for their valuable input during the course of my dissertation.

This thesis has benefited from the comments of the anonymous referees of my publications, especially the ones who reviewed my submissions to Journal of Networks and Journal of Mobile Information Systems.

In addition, I would like to express my appreciations to my students, my colleagues and staff especially Sue Peti and Marcy Curtiss who have made the department a great place to work over all those years.

On a personal note, I would like to express my gratitude to my parents Iqbal and Muhammad Bin Ahmed for their love and emotional support. Special thanks to my parents-in-law Sanjeeda and Mohammed Asif for support and encouragement. My special thanks to my sisters Fauzia, Farah, Saba, Hina, and Nausheen for love and support. My thanks to my bothers Khalid, Kashif, Yousuf, Raheem, Zahir, Ali, and Farooq for love and encouragement and to all my nieces and nephews for love.

Lastly and most importantly, I thank my best friend and wife Uzma for her moral support and for the incredible amount of patience she had with me in the last quarter of a century.
CHAPTER 1

INTRODUCTION

‘Excellent!’ I cried. ‘Elementary,’ said he.

—Dr. Watson in The Crooked Man by Sir Arthur Conan Doyle (1859-1930)

1.1 About this Work

The thesis studied the algorithmic issues in the design and analysis of parallel and distributed geometric algorithms and, Steiner trees and its applications. The thesis is based on two journal papers [1, 2] and twenty two conference proceedings, workshop papers, posters, etc. [3–24]. In particular, the thesis presents our results on the parallel Steiner trees, the parallel Voronoi Diagram, distributed range assignment, and distributed geometric routing. This work has already received considerable recognition in the research community, which is visible in citations in recently published scholarly works [25–29].

1.2 Motivation

The study of geometric algorithms and the Steiner tree problem is motivated by its central role in network design. We have chosen the Steiner tree in the context of geometric algorithms because it is one of the oldest and well established optimization problem and like other classical combinatorial problems such as traveling salesperson problem (TSP), they serve as a ground for discoveries. For example, the general concept of the “neighborhood composition”, which was introduced in the TSP context, we have used in the design of parallel Steiner tree algorithm. Besides, these optimization problems arise in situations in which we have to make discrete choices, and solving those requires finding an optimal solution among a finite number of alternatives. With these problems the difficulty are twofold. Not only are these problems too difficult to solve in polynomial time, but the quality of a solution may not be very attractive. For instance, in case of Steiner tree problem computing any feasible solution is not satisfactory, but the quality of solution is also very important. Several strategies have been proposed and applied to different optimization problems. Among those, local
search is very popular mainly because it gives rise to algorithms that are quite simple and robust. Simplicity and robustness come from the facts that these algorithms use some kind of greediness or priority based procedures. In general, a local search algorithm works as follows. It starts with an initial (current) solution and repeatedly replaces the current solution with a better one until no better solution can be found in its neighborhood. The main drawback of these techniques is that their search operations can be trapped in a local optimum. This is a clear indication that we need a technique for guiding these algorithms out of this “trap”. Following this theme, this thesis explores the possibility of extracting useful information from unsuccessful searches to improve the time complexity and quality of the final solution.

In the light of above arguments and with the explosion of parallel computers, parallel implementations of local search heuristics appear quite naturally to be a promising alternative to speedup the search for approximate solutions and to reach a better solution. Even though parallelism is not yet systematically used to speedup local search algorithms, parallel implementations are plentiful in the literature. A greedy randomized adaptive search procedure (GRASP) [30, 31] is a well-known multi-start paradigm. Most parallel implementations of GRASP are independent-threaded, multiple-walk strategies and based on the distribution of the iterations over the processors. Martins et al. [32] implemented a parallel GRASP for the Steiner problem in graphs based on the distance network [33]. The same approach was used in [34] for a preliminary parallel implementation of the GRASP for the Steiner problem in graphs. Canuto et al. [35, 36] used path-relinking as a post-optimization step to implement a multi-walk parallel GRASP algorithm for the prize-collecting Steiner tree problem. For local searches, they implement an independent multiple-walk strategy in parallel. Handa et al. [37] proposed a local search technique, called the neighborhood composition, and used it to reduce the processing time of the traveling salesperson problem. The neighborhood composition can be described as follows. Suppose a local search starts with an initial solution $s_0$ and tries to obtain an improved solution. Search ‘$i$’ suggests a substitution of a current solution $s_0 \in S$ with an $s$ to achieve a improved solution $s'$. Since the size of an instance is huge and search ‘$i$’ can explore only a restricted neighborhood of $S$ in a reasonable amount of time, many searches subsets (solutions spaces) do not overlap. Obviously, $s'$ can be further improved by replacing $s'$ with $s''$ using information in ignored searches for other neighborhoods. The basic idea of this technique is
to extract useful information from unsuccessful local searches to speedup the parallel local search algorithm for the traveling salesperson problem.

Similar to the parallel Steiner tree and Voronoi diagram, the fundamental role in the design of the wireless networking and mobile computing is the motivation behind the study of geometric routing algorithms with or without Steiner points (discussed in Chapter 6 and Chapter 7, respectively). Like all geometric/face routing algorithms [38–41], our geometric algorithm depends upon two factors namely, construction of a planar subgraph and a face traversal mechanism. The planar subgraph defines the underlying wireless connectivity graph while the face traversal defines a forwarding mechanism for the information packets from source to destination.

This thesis considers the unit disk graph model because an underlying assumption of the planar construction algorithm and the face traversal algorithm is that this computational model defines the connectivity among nodes in ad hoc wireless network. That is, any violation of the unit disk graph assumption results in failure either in the construction of planar subgraph or in the face traversal mechanism. As a result, geometric routing algorithm fails to deliver packet to its destination. We define the “failure” in context of geometric routing as the algorithm’s lack of ability to find a path for at least one pair of nodes between source and destination. Note that there are some studies [42] in which unit disk graph assumption is relaxed. The technique presented in those works is scalable only when certain restrictions were applied to the radius of communication nodes. Moreover, the technique involves virtual links.

In general, our geometric routing algorithm relies on three distinct components. First, our algorithm uses greedy forwarding paradigm. That is, a node under consideration sends a packet to its neighbor node that is geometrically closest to the destination node. Second, to traverse the face, our algorithm uses the right-hand rule and face changes. Note that an important aspect of the right-hand rule is that it must be supplemented with a rule that clearly defines when to change face. Our algorithm follows the suggestion of Bose et al. [38]. Third, the construction of planar subgraph. The primary reason of this construction is that geometric routing provably works correctly on a network graph that has no crossing links, i.e. a planar graph. In addition, a planar construction of subgraph eliminates crossing links to generate the structure that is required for the face traversal. In the literature, geometric routing algorithms construct planar subgraph using proximity structures from
computational geometry particularly, the *Gabriel Graph* (GG) [43], the *Relative Neighborhood Graph* (RNG) [44], or *Restricted Delaunay Graph* (RDG) [45]. The reason for the use of these graphs is that these proximity structures provably yield a connected, planar graph provided the connectivity among the nodes obeys the *unit disk graph* assumption. A *unit disk graph* is the intersection graph of a family of equal-radius circles, or of equal-radius disks in the Euclidean plane. Note that every induced subgraph of a unit disk graph is also a unit disk graph.

Taking account of all three components of geometric algorithms, we could generalize the overall structure of our work as follows.

| 1 | Construct the planar subgraph from the graph represented the given set of nodes. |
| 2 | A node sends packet to its geometrically closest neighbor until it reaches to the destination. |
| 3 | If there exists no such neighbor i.e., greedy traversal fails. |
| 4 | Then invoke the greedy (face) traversal algorithm (right-hand rule is implicit in this step) to recover from failure. |
| 5 | If face traversal come across a node closer to the destination. |
| 6 | Then recover from failure by falling back to greedy traversal i.e. Goto Step 2. |

**Algorithm 1.1**: Overall Structure.

In the remainder of this section, we discuss some major reasons, which lead to the failure in the construction of planar subgraph. From the practical viewpoint, the *unit disk graph* assumption fails two circumstances. Firstly, the node may estimate its position incorrectly. Secondly, the presence of obstacles, which causes the regularity in the communication ranges. These two practical circumstances cause the best geometric/phase routing algorithms [38–41, 46] in severe degrading in the performances, even responsible for failure in some instances. The deviations from ideal behavior can cause three kinds of failures in the construction of planar subgraph.

One of the most important consequences of the node’s incorrect estimation of its position is that it can leave unidirectional links in the derived subgraph [41], which is usually *Gabriel graph* or *relative neighborhood graph*. Particularly, At least one of the node $x$ (say) does not know the existence of at least one node $y$ (say) in the communication graph and node $x$ keeps link with some other node that should not be there in the resultant subgraph. On the other hand, other node removes the same link. It is easy to see that incorrect estimate of node’s position leads to these unidirectional links. Furthermore, these unidirectional links can produce an infinite loop during face traversal. Using the same line of arguments, we can show that the algorithm may leave the cross-links in the resultant subgraph. References [41, 47] discuss incorrect removal of links and leaving the cross-links in the subgraph in their simulation studies.
The presence of obstacles can also cause a link to be incorrectly removed from the resultant subgraph, which clearly results in a disconnected subgraph. As we have mentioned above, a disconnected resultant subgraph causes a face traversal failure. By failure in the traversal we mean that greedy traversal cannot reach certain nodes, since the resultant subgraph is partitioned. Again, a similar line of arguments can show that crossing links in the resultant subgraph can cause persistent routing failures. In this case, the failure is usually due to the infinite loop.

1.3 Notations and Background Information

The notations of this work are compatible with "The Steiner Tree Problem" [48] in Annals of Discrete Mathematics. Whenever we need additional notations, we tried to extend the notation of [48] instead of coming up with entirely the new ones.

The primary sources for the definitions and background information of the Steiner trees, the Voronoi diagram, the range assignment, and the geometric routing are the references [48–50]. The secondary sources are [51–58].

1.4 Structure of the Thesis and Main Contributions

The Chapter 2 and Chapter 3 deal with preliminaries and computational models, respectively.

The main contribution of the Chapter 4 is an efficient and implementable algorithm based on a greedy randomized adaptive search procedure for Euclidean Steiner tree problem (ESTP) on the Server-Client model of computation. We introduce an algorithm for an event-driven simulation of the problem on the server-client model. This algorithm acts like a ground or arrangement for the proposed parallel algorithm works on which we can implement/simulate the proposed scheme for the ESTP. We perform an depth analysis of execution time, from practical and implementation point of view, of execution time of ESTP. Also, we examine some consequences of time delays of processors (server-clients) on the execution time. Our contribution is the computation of the expression for the optimal number of client processors required by the parallel algorithm for optimal efficiency. The compact mathematical expression can be used confidently to simulate or implement the parallel Steiner algorithm on the server-client model.

The main contribution of the Chapter 5 is the parallel construction of a Voronoi diagram for a set of $n$ points in the Euclidean plane on the hypercube model of computation.

The Chapter 6 discusses the problem of introducing relay nodes in the given set of original
nodes and assigning transmitting ranges to the nodes so that the cost of range assignment function is minimal in the communication graph over all connecting range assignment functions. The main contribution of this chapter is the $O(n \log n)$ algorithm (Add-Node) to add relay nodes to the wireless network. The output of the algorithm is the minimal Steiner tree on the graph consists of terminals (original) nodes and the relay (additional) nodes. Our proposed scheme does not assume that the communication graph to be a unit disk graph. We proposed a 2-approximation algorithm to assign transmitting ranges to nodes (original and relay) in the communication graph. Our approximation scheme uses the output of the Add-Node algorithm and output the range assignments to the nodes.

The main contribution of the Chapter 7 is an algorithm to extract a connected, planar subgraph of the underlying wireless connectivity graph. This chapter also presents a geometric routing algorithm on the planar, subgraph previously extracted.

Finally, some concluding remarks collected in the Chapter 8.
CHAPTER 2

PRELIMINARIES

In this chapter, we look at the basic definitions, notations, and ideas, in a very broad way to set the stage for later chapters. The chapter is introductory and contains a relatively large number of definitions and notations. The goal of the chapter is to provide a reasonably good grasp of the terminologies of the thesis.

2.1 Euclidean Steiner Problem

The Euclidean Steiner problem (ESP) or Euclidean Steiner minimal tree (ESMT), problem asks for a shortest tree spanning a given set $N$ of $n$ terminals in the plane. Contrary to the minimum spanning tree (MST) problem, connections in minimum Steiner tree (SMT) are not required to be between the terminals only. Additional intersections, called Steiner points, can be introduced to obtain a shorter spanning tree. The ESP problem is as follows. For a given set $N$ of points in the plane, construct a minimal length tree, which connects terminals in $N$. In order to minimize the total length of the tree, additional points $S = s_1, \ldots, s_m$ can be used. Here, the distance metric is the $L_2$ metric function, known as the Euclidean metric.

![Steiner Points](figure21.png)

Figure 2.1: Steiner tree for six nodes or terminals.

2.1.1 Relevant Steiner Tree Properties

We shall mention here only some basic properties of SMTs needed in the paper. The reader referred to [48] for a comprehensive survey. (1) Steiner points are incident with exactly three edges

\footnote{In Hwang et al. [48], an SMT is called a minimum Steiner tree. We will use SMT to avoid the acronym MST, which usually means minimum spanning tree.}
making 120° with each other (angle condition). (2) SMTs for \( n \) terminals have at most \( n-2 \) Steiner points. (3) SMTs are union of full Steiner trees (FSTs) terminals spanned by an FST have degree one. FSTs have two Steiner points less than they have terminals. If two FSTs share a terminal \( N \), then the two edges incident make at least 120° with each other. Therefore, no terminal can be in more than three FSTs. In other words, each terminal \( N_i, 1 \leq i \leq n \), has at most degree 3 i.e., \( \text{deg}(n_i) \leq 3 \). (4) It has been conjectured that the MST is no more than \( \frac{2}{\sqrt{3}} \) times as long as the minimum length ESMT [59]. Polak has shown that this conjecture is true for \( n = 4 \). Graham and Hwang [60] showed that the lower bound on the SMT/MST \( \geq \frac{1}{\sqrt{3}} \). The problem has been shown to be NP-Complete in [61].

2.1.2 Relevant Steiner Tree Results

Now, we present some classical results regarding 3-terminal and 4-terminal Euclidean Steiner problem (ESP) and discuss their incorporation into the framework of our algorithms.

In case of three terminals, if one of the angles of \( \triangle abc \) is at least 120°, then the Steiner tree consists of simply the two edges subtending the obtuse angle. Therefore, assume that all internal angles of \( \triangle abc \) are less than 120°. The addition of an additional point (or Torricelli point) in \( \triangle abc \) is depicted in Figure 2.3 [48].
Lemma 2.1. An additional point in the plane, the sum of whose distances from three given points is minimal, can be located if and only if all internal angles of a triangle are less than $120^\circ$.

![Figure 2.4: The Quadrilateral abcd.](image)

In case of four terminals, consider the quadrilateral abcd in the Figure 2.4. Points $e_{ab}$ and $e_{cd}$ form equilateral triangles with $(ab)$ and $(cd)$ respectively. By Du et al. [62] the necessary and sufficient condition for the existence of FST in that the quadrilateral abcd must be convex. Weng [63] showed that convexity of the quadrilateral abcd (see Figure 2.4) is implied by following two conditions.

1. $\angle edaab$, $\angle eabcd$, $\angle edaab$, and $\angle eabca$ are all less than $120^\circ$.
2. $\angle doc < 120^\circ$.

An important insight of Polak’s [64] paper on existence of SMT of four terminals can be stated as follows.

Lemma 2.2. If four terminals $a$, $b$, $c$, and $d$ form a convex quadrilateral, then SMT exists for those terminals.

![Figure 2.5: The addition of two additional points.](image)

The Figure 2.5 illustrates the addition of 2 additional points in quadrilateral. For proof of the fact that (a), (b), and (c) in this figure are sufficient to define the optimal Steiner tree for 4 terminals see Gilbert and Polak [59].

2.2 Voronoi Diagram

Let $P = \{ p_1, p_2, \ldots, p_n \}$ be the set of points in the Euclidean plane. Also, let $H(p_1, p_2)$ be the half-plane and defined as the set of points equidistant to $p_1$ and $p_2$, then $V(p_1, P) = \cap_{p_j \in P \setminus p_1} H(p_i, p_j)$
be the Voronoi region of \( p_i \). The Voronoi region, \( V(p_i, P) \), is convex and its interior is the locus of points closer to \( p_i \) than to any other point. Therefore, \( V(p_i, P) = \{ q \in E^2 : |p_i q| \geq |p_j q|, \forall p_j \in P \setminus p_i \} \).

Let \( P(p_i, P) \) denote the boundary of \( V(p_i, P) \). The union of these boundaries for all points in \( P \) forms the Voronoi diagram for \( P \), denoted by \( VD(P) \). The \( k \)-th order Voronoi diagram \( VD_k(P) \) is a partition of the plane into regions \( V(P_k, P) \), where \( P_k \subseteq P \) and \( |P_k| = k \). The interior of \( V(P_k, P) \) is the locus of points closer to every point in \( P_k \) than to any point in \( P \setminus P_k \). Therefore, \( V(P_k, P) = \{ q \in E^2 : |p_i q| \leq |p_j q|, \forall p_i \in P_k, \forall p_j \in P \setminus P_k \} \). Note that \( VD_1(P) = VD(P) \). The Voronoi diagrams of all orders up to \( K \)-th order can be determined in \( O(K^2 n \log n) \) time [65].

![Figure 2.6: The Voronoi Diagram of eleven planar points.](image)

Let \( N \) be a set of \( n \) planar points. The Voronoi diagram partitions the plane into regions such that each region contains exactly one point. The region of a point \( n_i \in N \) consists of all points closer to \( n_i \) than to any other points on the plane. The region of a point \( n_i \in N \), \( 1 \leq i \leq n \), is called the Voronoi cell or Voronoi polygon and denoted as \( VP(n_i) \). The vertices of these polygonal regions are called Voronoi vertices and the polygonal boundaries i.e., edges of the regions, are called Voronoi edges. The collection of Voronoi polygons \( VP(n_i) \) for each \( n_i \in V \) is called Voronoi diagram and often denoted as \( VD(N) \). For further discussion on the properties of the Voronoi diagram, see [24,66].

2.3 Delaunay Triangulations

The straight-line dual of the Voronoi diagram for the set \( N \) of \( n \) point is a triangulation of \( N \), known as Delaunay triangulation and denoted by \( DT(N) \). Also, defined as the unique triangulation in which the circumcircle of each triangle does not contain any other point in its interior. Delaunay Triangulations can be computed in \( \Theta(n \log n) \) time and contains at least one minimum spanning tree (MST) for the given set \( N \). The MST for \( N \) can be computed in \( \Theta(n) \) time once Delaunay
triangulation of $N$ is given \cite{67}.

Figure 2.7: Delaunay Triangulations (dual of Voronoi diagram of Figure 2.6).

For a given set $N$ of $n$ points, *Delaunay Triangulation*, denoted by $DT(N)$ is defined as the unique triangulation in which the circumcircle of each triangle does not contain any other points of $N$ in its interior. *Delaunay Triangulations* can be computed in $\Theta(n \log n)$ time and contains at least one minimum spanning tree (MST) for a given set $N$. Note that MST for $N$ can be computed in $\Theta(n)$ time once the *Delaunay triangulation* of $N$ is given \cite{67}.

The *Delaunay triangulation*, $DT(N)$, is the straight-line dual of Voronoi diagram $VD(N)$. Each Delaunay face (triangle) corresponds to a Voronoi vertex. The interior of each Delaunay triangle of $DT(N)$ contains no point $n_i \in N$. The Voronoi diagram and the Delaunay diagram can be constructed in $O(n \log n)$ time \cite{68}. Reader may consult \cite{49} for further information.

Figure 2.8: Delaunay Triangles abc and bcd correspond to Voronoi vertex v1 and v2, respectively. Note that a triangle abc belong to the Delaunay triangulation, if its circumcircle disk(a, b, c) does not contain any other point in its interior.

**Lemma 2.3.** Given a set $N$ of $n \geq 3$ points. The $DT(N)$ has $E = 3(n - 1) - |\partial CH(N)|$ number of
edges and $T = 2(n - 1) - |\partial CH(N)|$ number of triangles, where $|\partial CH(N)|$ is the number of points on the boundary of the Delaunay triangulation of the point set $N$.

**Lemma 2.4.** $DT(N)$ contains at least one MST for $N$. The MST for $N$ can be determined in time $O(n)$ from $DT(N)$ [67].

The $k$-localized Delaunay graph, $LDel^{(k)}(N)$, on the nodes set $N$ has unit Gabriel edges and edges of all $k$-localized Delaunay triangles. Li et al. [69,70] defined the $k$-localized Delaunay triangle as the triangle if the interior of the circumcircle of the triangle does not contain any node of $N$ that is a $k$-neighbor of any of three nodes of the triangle. Furthermore, all edges of the triangle have no more than a unit length. Li et al. [69,70] have shown that the graph $LDel^{(1)}(N)$ may contain some intersecting edges but the graph $LDel^{(2)}(N)$ contains no intersecting edges i.e. planar. Hence the following lemma.

**Lemma 2.5.** For any $k \geq 2$, The $k$-localized Delaunay graph, $LDel^{(k)}(N)$, is a planar graph [69,70].

### 2.4 Gabriel Graph

The *Gabriel graph*, denoted by $GG(N)$, of set $N$ is defined by the graph in which $uv$ is an edge of $GG(N)$ if, and only if, the circle having $|uv|$ as a diameter is an empty circle, that is, if and only if it contains no points of $N$ in its interior. $GG(N)$ can be computed in $\Theta(n \log n)$ time by removing edges $DT(N)$ edges not intersecting their Voronoi edges. Note that $GG(N)$ is a subgraph of $DT(N)$ and contains at least one MST for $N$. For details reader may consult [49]. The unit Gabriel graph, $GG(V)$, consists of all edges such that $|uv| \leq 1$ and the circle having diameter $uv$ does not contain any vertex from $V$ and the lengths of edges are not more than unity.

![Figure 2.9](image)

**Figure 2.9:** Gabriel Graph (GG) contains an edge $uv$ if the disk with $uv$ as diameter is empty of other points.

### 2.5 Relative Neighborhood Graph

The *relative neighborhood graph*, denoted by $RNG(N)$, is defined as a geometric graph in which $RNG(N)$ has an edge between $n_i$ and $n_j$ if and only if $d(n_i, n_j) = \min_{k \neq (i,j)} \max(d(n_i, n_k), d(n_j, n_k))$. 
RNG(N) can be computed in $\Theta(n \log n)$ time. Furthermore, RNG(N) is a subgraph of DT(N) and contains at least one MST for N.

Figure 2.10: Relative Neighborhood Graph (RNG) contains an edge $uv$ if the lune is empty of other points.

2.6 Geometric Graph

This thesis assumes that the underline graph is a geometric graph. A geometric graph $G = (N, E)$ is a graph in the plane so that its vertices are points and its edges are straight-line segments connecting pair of these points. Recall, a geometric graph need not be planar. Note that when it is clear from the context, we refer to the element of $N$ interchangeably as host, nodes, or vertices. We represent sensors with points, relay sensors with additional points or Steiner points, and sensors transmission radius or distance between two sensors with edge length or Euclidean distance.

2.7 Range Assignment

A range assignment for the set $N$ on network nodes is a function RA that assigns to $u \in N$ a transmitting range $RA(u)$. Formally, let $N$ be a set of nodes in the Euclidean space $R^d$, $d=2$. The problem is to determine a range assignment function RA such that the corresponding communication graph is strongly connected and Cost(RA)=$\sum_{u \in N}(RA(u))^\alpha$ is minimum over all connecting range assignment functions, where $\alpha$ is the distance-power gradient.

2.8 Neighborhood

The open neighborhood $\mathcal{N}(v)$ of the vertex $v$ consists of the set of vertices adjacent to $v$, that is, $\mathcal{N}(v) = \{w \in N : vw \in E\}$ and the closed neighborhood of $v$ is $\mathcal{N}[v] = \mathcal{N}(v) \cup \{v\}$. For a set $N$, the open neighborhood $\mathcal{N}(N)$ is defined to be $\bigcup_{v \in N}\mathcal{N}(v)$, and the closed neighborhood of $N$ is $\mathcal{N}[N] = \mathcal{N}(N) \cup N$. 
2.9 Right Hand Rule

The key to the Face Routing algorithm [46] is the right hand rule. The right-hand rule [71] states that when arriving at node $x$ from node $y$, the next edge traversed is the next one sequentially counterclockwise about $x$ from edge $(x, y)$. It is known that the right-hand rule traverses the interior of a closed polygonal region (a face) in clockwise order and traverses an exterior region in counterclockwise edge order.

2.10 Communication Graph

The communication graph defines the set of wireless links that the nodes can use to communicate with each other. Based on the above discussion, the existence of the edge (link) between two nodes depends on the distance between the nodes, transmission range assignment, and environmental conditions i.e., distance power gradient, $\alpha$. Given a transmitting range assignment, we define communication graph $G$ such that directed edge $(u, v) \in G$ if, and only if, $\text{dist}(u, v) \leq r(u)$ i.e., Euclidean distance from $u$ to $v$ is at most $r(u)$.
CHAPTER 3

COMPUTATIONAL AND MATHEMATICAL MODELS

3.1 Introduction

The Computer science is an extraordinarily diverse field, topics range from hardware design to programming. Nevertheless, like any other science, there are some common underlying principles and algorithmic issues. To study these algorithmic issues, we must choose (or construct) the appropriate abstract model of computer and computation. Our constructed or chosen abstract model should represent the algorithmic features that are common to both hardware and software. The argument is still valid even when the model is too simple to be right away relevant to real-world situations; however, the insights we gain from algorithmic studies on those simple models provide the foundations for working on real-world computer and computational problems. In fact, the usefulness of a computer science (or any scientific discipline) is often dependent on the existence of simple theories and laws, which in turn depend on the simple, yet powerful model.

For a given problem, different computational models have lists of advantages and disadvantages. The network model seems to be well suited for some problems. For example, construction of Voronoi diagram seems natural on the hypercube model of computation. As the chapter shows that network model, in particular, the hypercube resolves both computation and communication issues quite efficiently. Because of the characteristics of hypercube model, we are able to design very efficient $O(lg^3n)$ algorithm. In spite of these characteristics, the analysis of algorithms on network models, including hypercube, is quite difficult. The reason for this difficulty is that the analysis depends heavily on a particular topology under consideration. For instance, the analysis of Voronoi diagram on hypercube model is radically different from the star topology. We can make a similar argument for the parallel Steiner tree problem. These reasons indicate that the shared-memory model is more suitable algorithmic model for parallel algorithms. In addition, we believe that the shared memory model is more suitable for the general presentation of parallel algorithm. Distributed shared memory (DSM) is the implementation of the shared-memory abstraction on a multicomputer architecture. Nitzberg and Lo [72] and Stumm and Zhou [73] algorithm for implementing distributed shared memory.
In this work, we are mainly interested in problems and their algorithmic solutions. In this framework, we use the model as a tool for thinking about the problem and expressing its solution algorithmically. In general, the computational models considered in this work are not necessarily linked to any real computer. We emphasize, the fundamental reason for models in this work is to understand the spirit of the computation. We strongly agreed with Professor Akl\footnote{Personal communication with Professor Selim G. Akl, Queens University, Canada.} and paraphrase that (theoretically) a model is simply a framework for examining computational problem, gaining an insight into the problem’s (mathematical and geometrical) structure, and developing (algorithmic) solutions to the problem under consideration.

3.1.1 Chapter Guide

Section 3.2 presents the Server-Client model of computation for the study of Steiner tree.

Section 3.4 discusses the hypercube interconnection network model for the study of Voronoi diagram.

Section 3.3 presents the Long-Distance model for ad hoc wireless networking.

3.2 Server-Client Paradigm

Server-Client paradigm describes a pattern of work sharing. A server process offers a service to the client processes. Server-client is the most common paradigm of distributed computing at present. However, the paradigm can be used in the parallel domain since most applications involve multiple clients as opposed to multiple servers. The first process, the server, manages some resource, for instance a MST. It offers a service (for instance, upgrade in solution), which can be used by other processes. For example, the service is to enhance or upgrade the solution. The server, once started, runs permanently over a long period. It is passive in that it does nothing but wait for a client request. The client, on the other hand, is a process that needs the service in order to accomplish its work. Therefore, it sends a request to the server, in which it asks for execution of a concrete task that is defined by the service. For example, the client may want to retrieve a particular solution space (local solution or simply sub-solution) from the Server-Solution-Space (global solution). Upon receiving the request, the server becomes active, carries out the task of updating the global solution, and sends the result back to the client. Thereafter, the server becomes passive again and waits for the next request from the same client (to enhance the solution just received) or from a different client.

By definition, Client-Server computing does not include parallelism, but neither does it exclude
it [74]. To achieve parallelism, the client arranges multithreading so that a processor can switch to another task while waiting for an outstanding request. In essence, we are combining the client-server paradigm with parallelism. The result of this setup is improvement in the performance.

The Figure 3.1 depicted the fundamental arrangement of algorithmic and machine components that we have used in this chapter to implement the parallel Steiner tree algorithm and improve its performance. Note that this setup is based on the Unix Network programming by W. Richard Stevens [75].

![Server-Client Computational Model](image)

**Figure 3.1: Server-Client Computational Model.**

### 3.3 Long-Distance Model of Computation

The thesis assumes the long-distance model [76], which abstracts many characteristics of the environment. In this model, the average long-distance path loss is proportional to the separation distance between transmitting node $v$ and receiving node $t$ raised to a certain exponent, $\alpha$, which is called the distance power-gradient. The value of $\alpha$ depends on the environmental conditions and it has been experimentally evaluated in many scenarios [77]. Typically, nodes are located in a two-dimensional Euclidean space. These nodes are connected by wireless links that are induced by their energy level. According to the long-distance model, in the given region, each node $u$ transmits a signal with power $P(u)$, which is received by another node $v$ such that $\text{dist}(u,v)^\alpha \leq P(u)/\gamma$, where $\text{dist}(u,v)$ is the Euclidean distance between nodes $u$ and $v$, $\alpha \leq 1$ is the distance power gradient, and
\[ \gamma \leq 1 \] is the transmission quality parameter. Without loss of generality, in the thesis, we assume \( \gamma = 1 \). The radio coverage region in this model is a disk of radius at most \( r(u) \geq (P(u)/\gamma)^{1/\alpha} \) centered at the transmitter \( u \). Since \( \gamma = 1 \), thus we have \( r(u) \geq (P(u))^{1/\gamma} \). The value \( r(u) \) is the transmission range of transmitter \( u \), i.e., the maximum distance at which \( u \) can transmit in one hop with power \( P(u) \).

![Figure 3.2: Transmitting Range in two-dimension. The transmission range has radius \( r \) and it is centered at the node \( u \).](image)

The thesis considers the standard model for ad hoc networks where all nodes have the same transmission range. That is, in this work we model an ad hoc network as a geometric graph. A graph \( G \) is defined as a pair \((N, E)\), where a set \( N \) representing the nodes of the network and a set \( E \subseteq N^2 \) denotes the set of edges, connecting the nodes in \( N \). Furthermore, there exists an edge between two nodes \( u \) and \( v \) if, and only if, the nodes \( u \) and \( v \) are within mutual transmission range. In the remainder of the thesis we will refer to the elements of \( N \) alternately as host, nodes, or vertices. Since, we are assuming that all nodes have equal broadcast ranges. This leads to the definition of the unit disk graph \((UDG)\). Let \( N \) be a set of points in the plane. Then the \( UDG \) is a geometric graph that contains a vertex for each element of \( N \). An edge \((u, v)\) is present in \( UDG(N) \) if, and only if, \( dist(u, v) \leq 1 \) (this can be accomplished by scaling the coordinate system), where \( dist(u, v) \) denotes the Euclidean distance between \( x \) and \( y \). Now if we combine the above definitions of graph and unit graph; and generalize the statement, we have a graph \( G \) where each node has a coordinate in 2-dimensional space. More formally, a 2-dimensional embedding of \( G \) is a coordinate function \( \text{coord} \): \( N \rightarrow \mathbb{R}^2 \) of the nodes. A graph \( G \) is a unit disk graph \((UGD)\) if it has an embedding such that the Euclidean distance \( \text{dist}(\text{coord}(v), \text{coord}(u)) \leq 1 \) if, and only if, \( \{u, v\} \in E \).

**3.4 Hypercube Model of Computation**

The hypercube is an important topology that has received tremendous attention in the theoretical literature [78]. In [79], Foster describe various useful algorithms on a hypercube and a parallel
algorithm template in addition to a detailed description of communication structure of hypercube. The readers may refer to [80] for basic communication operations on hypercube. The $d$-dimensional hypercube has $n = 2^d$ nodes and $d2^{d-1}$ edges. Each node corresponds to a $d$-bit binary string. The hypercube connects the two nodes with an edge if and only if binary strings differ in exactly one bit. As a result each node is incident to $d = \log_2(n)$ other nodes. Let $n = 2^d$ processors $P_0, P_1, \ldots, P_{n-1}$ be available for $d > 1$. Further, let $i$ and $i^b \leq n - 1$, whose binary representation differ only in position $b$, $0 \leq b \leq d$. A $d$-dimensional hypercube is formed by connecting each processor $P_i$, $0 \leq i \leq n - 1$, $P_i^b$ links, for all $0 \leq b \leq d$.

![Figure 3.3: The eight nodes hypercube.](image)

The recursive property of an hypercube is that $k$ edges in a hypercube form a perfect matching for each edge $k$, $1 \leq k \leq \log(n)$. Moreover, removal of $k$ edges for any $k \leq \log(n)$ results in two disjoint subhypercubes each containing $n/2$ nodes. On the other hand, an $n$-node hypercube by simply connecting two corresponding $i^{th}$ nodes for $0 \leq i \leq n/2$. Additional properties of hypercube are that it has low diameter, $\log(n)$, and high bisection width, $n/2$. Moreover, hypercube is node and edge symmetric.

3.5 Bibliographical Remarks

The theory of computation is incomplete without the models of computation. The readers are referred to Hopcroft and Ullman [81] and Davis and Weyuker [82] for the general discussions of various computational models. Our interpretation and analysis of computational model is primarily based on Akl et al. [83]. Other references we used in our investigation are Cormen et al. [84] and JaJa [85]. Interconnection network computational model for parallel computer are studied in Hawng and Briggs [86], Reed and Fujimoto [87], Stone [88, 89], Varma and Raghavendra [90], and
Wu and Feng [91]. Our description of hypercube model of computation is based on the Leighton [92]
and the analysis of our proposed algorithm on hypercube is based on Fox et al. [93] and Ranka
and Sahni [94]. The in depth discussion of the interconnection networks and their graph-theoretic
properties and routing and broadcasting algorithms, is provided in Stojmenovic [95]. Our discussion
and interpretation of the server-client model of computational is based on [75] and [74].
CHAPTER 4

PARALLEL STEINER TREE

4.1 Introduction

Historically, the first Steiner problem to be considered was a geometric Steiner problem in a graph. In fact, the Steiner problem in network is a combinatourial version of the Euclidean problem, which is the theme of this chapter. That is, given a weighted graph $G = (V, E, w)$ in which a subset of vertices are identified as terminals $N$, find a minimum-weight connected subgraph that includes all the terminals. Two special cases of the Steiner problem in a network are polynomially solvable. If $|N| = 2$, the problem reduces to a shortest path problem and if $N = V$, the problem reduces to the minimum spanning tree problem (MST). Interested readers are referred to [96]. The parallel computation allows us to look at the problems that have previously been impossible to solve (calculate) accurately, as well as to compute in reasonable time. Due to this reason, we started to look at a parallel algorithm for the minimum Steiner tree (SMT) problem. The published results in the references [3, 5, 7, 9, 15, 17, 20, 97] are the basis of the chapter.

While the fundamental approach is based on the neighborhood composition [37] and proximity structure [98], our proposed algorithm diverges in a significant aspect: the way algorithm finds the additional points called Steiner points in parallel by first dividing the neighborhood structure into geometrical subgraphs. Furthermore, the implementation details are very different due to the fact that the Steiner tree problem required to include Steiner points. Zachariasen et al. proposed a sequential local search algorithm for Steiner tree problem in [98]. The approach is based on a listing of the full Steiner trees, FSTs, in the preprocessing phase. The list of full Steiner trees is the solution space. For neighborhood structure, they used the well-known structures from computational geometry, such as Voronoi diagram (VD), Delaunay triangulation (DT), Gabriel graph (GG), relative neighborhood graph (RNG), etc. In a straightforward parallel implementation of local search algorithm for Steiner tree problem, the given set terminals is divided into several subsets (subsolution spaces), which are distribute over the processors. Each processor performs a sequential
search for better solution, defined by the condition of the terminals appearing in the solution, in its neighborhood. Before the next local search starts, acknowledge the result of one search (that looks promising at that time) and ignore the results of all other searches despite of the fact that they would contain useful information for further searches. Our experiments showed that this kind of straight forward parallel implementation will trap the local searches into a local minimum and severely damage the quality of the final solution.

4.1.1 Our Contribution

The main contribution of this chapter is a proficient and implementable algorithm based on local search algorithm for Euclidean Steiner problem (ESP). This work parallelizes the well-known GRASP paradigm and efficiently combines the sequential approach based on computational geometry structure [98] with the concept of neighborhood composition [37] to speedup the processing time of Euclidean Steiner problem. The secondary contributions are an algorithm for an event-driven simulation algorithm for the problem on the server-client model, analysis technique for the algorithm's execution time and the mathematical expression for the optimal number of processors required by the parallel algorithm for optimal efficiency.

4.1.2 Chapter Guide

Section 4.2 presents the outline of a parallel local search algorithm for Steiner tree problem in the Euclidean plane.

Section 4.3 states the formal parallel Steiner tree algorithm, which is based on the relative neighborhood graph (RNG). We will perform complexity analysis and establish the correctness of the proposed algorithm.

Section 4.4 deals with the time delays among server and client processor’s for the parallel Steiner tree algorithm and derives mathematical expressions for the optimal number of processors required by the proposed algorithm. Here we deal with two cases namely, constant and variable time delays among processors.

Section 4.5 gives a general technique to simulate the proposed algorithm. Here we will discuss many theoretical issues concerning simulation of the algorithm such as halting condition, simulation variables etc. In addition, we will present the overall strategy of the simulation.

Section 4.6 presents the formal simulation algorithm.

Section 4.7 gives the illustrative example of the server queue simulation algorithm.
Section 4.8 presents the summary of the chapter.

4.2 Overall Strategy

The proposed parallel Steiner local search algorithm is based on [37] and [98] and uses the server-client paradigm (generally known as master-slave paradigm). It has two phases, namely search phase and upgrade phase. These two phases are accomplished by server and client processors. Server processor does the upgrading in the solution while searches are accomplished by client processors. The algorithm is based on the decomposition of the solution space. The server processor reduces the ESTP to a selection problem by constructing a list of full Steiner trees \( F = \{f_1, f_2, \ldots, f_m\} \), where \( m = O(n) \) is the number of FSTs [98]. The server processor divides the ESTP problem by generating subsets of terminals by enumerating all connected subgraphs of proximity structure (for example, RNG) up to \( k = 5 \) terminals. Then, it generates lists of full Steiner trees (FSTs) associated with each subgraph. The subgraphs and lists of FSTs associated with each subgraph are distributed over the client-processors.

In a local search phase, a local minimum in the neighborhood of the constructed solution is sought. Each client-processor searches a solution space i.e., list of FSTs on a different subgraph, defined by condition the vertices appearing in the solution. The best solution over all searches is kept as the result by client processor. When a client finds a better solution in its neighborhood, it computes the enhanced solution from its current solution and newly calculated one, and sends the enhancement (not the best solution) to the server processor. We define enhancement as the difference between the original solution and the best solution so far. After the enhancement has been received, the server tries to apply the enhancement to its current solution. If this tryout is successful that is, it actually improves the overall solution, the server sends a new copy of its solution to the client. Otherwise, server processor updates the segment of the solution spaces assigned to the client processor. Since the current solution of the client may be completely different from the server’s current solution.

The advantage of this method is that it has no communication overhead (theoretically) since it does not require synchronization of client processors. That is, each client independently searches its neighborhood while other clients communicate with server.
4.3 Parallel Algorithm

This section presents the parallel algorithm, based on the [98] and [37], for Euclidean Steiner tree problem. For the ease of presentation, we consider only relative neighborhood graph in the following algorithm. The following algorithm is based on the Gabriel versions’s algorithm presented in [15].

Algorithm 4.1: Algorithm Server (UPGRADE)

1. For given set $P$ of $n$ points, construct $RNG(P)$
2. Set Server solution to $MST(P)$.
3. Generate subsets of terminals by enumerating all connected subgraphs of $RNG(P)$ up to $k=5$ terminals i.e., $RNG_i(P_k)$ for $1 \leq i \leq n$ and $1 \leq k \leq 5$.
4. Generate full topologies for each $RNG_i(P_k)$.
5. Send server solution, $\sigma_{server} = MST(P)$ and $RNG_i(P_k)$ to each client[i]. i.e., client[i] gets MST(P) and $RNG_i(P_k)$, where $i$ is the number of clients.
6. WAIT.
7. Enhancement in the $\sigma_{server}$ solution received from client[i], $1 \leq i \leq n$.
8. Check if Enhanced solution $\sigma_{client}$ received from client[i] is consistent with current server solution, $\sigma_{server}$.
9. If “yes” improved current server solution, $\sigma_{server}$, based on the received client’s enhancement.
10. Generate subsets of terminals by enumerating subgraphs of $RNG_{\sigma_{server}}$.
11. Send the current server solution, $\sigma_{server}$ and new partitions (subgraphs) $RNG_i$ to the client[i] from which the enhancement is received and GOTO Step 6.

Algorithm 4.2: Algorithm Client (SEARCH)

1. Receive the initial solution $\sigma_{server} = MST(P)$ and $RNG_i(P_k)$ where $1 \leq k \leq 5, 1 \leq i \leq n$ and set client solution $\sigma_{client} \leftarrow MST(P)$.
2. A solution $s' \in S$ has $s'_i = 1$, iff $F'_i$ is selected, $1 \leq i \leq n$.
3. Compute the Enhancement and send enhancement $\sigma'$ to the server where $Solution_{Enhanced} = Solution_{New} - Solution_{Old}$.
4. WAIT.
5. When received a $\sigma_{server}$ and $RNG_i(P_k)$ for neighborhood $N_i$ from the server, replace $RNG_i(P_k)$ and set $\sigma_{client}[i] \leftarrow \sigma_{server}$.
6. GOTO Step 2.

4.3.1 Complexity Analysis

The relative neighborhood graph can be constructed in time $O(n \log n)$ since it is an easily identified subgraph of the Delaunay triangulation. Zachariasen et al. [98] showed that a linear list of FSTs can be constructed in $O(n \log n)$ time. FSTs spanning two terminals in an SMT must belong to $MST(P)$ so these $n-1$ FSTs can be found in $O(n \log n)$ time since $MST(P) \subseteq RNG(P)$. We generate subsets of terminals by enumerating all connected subgraphs of $RNG(P)$ up to 5 terminals. The choice of 5 for a maximum number of terminals is based on the results in [99] and [98]. Since for a solution $x \in X$, a neighbor in $N(x)$ is constructed by flipping the value of one entry in $x$, this gives a total of $m$ neighbors, each neighbor is evaluated in time $O(m)$, where $m$ is the number of FSTS.
Thus, evaluating the whole neighborhood takes $O(m^2)$. Since $m = n - 1$, it is reasonable to assume $O(n^2)$. Therefore, the algorithm requires time $O(n^2)$ where $O(n)$ be the size of the neighborhood (in the worst case). Note that Zachariasen et al. proposed $O(n \log n)$ sequential algorithm in [98].

Following lemmas are based on [15].

**Lemma 4.1.** In the algorithm, the parallel time complexity of evaluating neighborhood is $O(n^2 \log n) + O(\log \left(\frac{n}{\log n}\right))$.

*Proof.* From above discussion, we know that the time complexity to evaluate an entire neighborhood with one processor is $O(n^2)$. With $p$ processors, our algorithm divides the neighborhood into $\frac{n}{\log n}$ subgraphs i.e., $RNG_i(P_k)$ for $1 \leq i \leq n$ and $1 \leq k \leq 5$, that contains $\log n$ solutions. Therefore, its time complexity is $O((n^2 \log n) + (\log \left(\frac{n}{\log n}\right)))$. \qed

**Lemma 4.2.** The parallel time complexity of the algorithm is $O(n^2 \log n + \log n \log \left(\frac{n}{\log n}\right))$.

*Proof.* From Lemma 4.1 we know the time complexity of evaluating the neighborhood is $O(n^2 \log n) + O(\log \left(\frac{n}{\log n}\right))$ and $O(n)$ be the neighborhood size for instance of size $n$. Let $t_p$ be the time required by an algorithm with $p$ processors (server + clients processors). Then the time required by the algorithm to evaluate neighborhood of size $O(n)$ using one processor is

\[
t_1 = O((\text{Neighborhood evaluation time})(\text{neighborhood size})) = O((n^2 \log n + \log \left(\frac{n}{\log n}\right)) \times n) = O(n^3 \log n + n \log \left(\frac{n}{\log n}\right))
\]

From Lemma 4.1, we know that the number of processors are $p = \frac{n}{\log n}$. Therefore,

\[
t_p = O((\text{Neighborhood evaluation time})(\text{neighborhood size})) = O((n^2 \log n + \log \left(\frac{n}{\log n}\right)) \times \log n) = O(n^2 \log^2 n + n \log n \log \left(\frac{n}{\log n}\right))
\]

Hence, the parallel time complexity of the algorithm is $O(n^2 \log^2 n + \log n \log \left(\frac{n}{\log n}\right))$. \qed

Kruskal et al. [100] introduced a class $EP$ of efficient parallel algorithms, in which they measured the performance of parallel algorithms relative to that of sequential algorithms. $EP$ is defined as
follows. Let the time complexity of a sequential algorithm and a parallel algorithm using \( p(n) \) processors for instances of size \( n \) be given by \( t(n) \) and \( T(n) \), respectively. An algorithm is polynomially fast and has constant efficiency, if \( T(n) = O(t(n)^\epsilon) \) with \( \epsilon < 1 \) and \( T(n) \times p(n) = O(t(n)) \). The class that contains these algorithms is called \( EP \). Using the definition of \( EP \) class, we established the following.

**Lemma 4.3.** The algorithm belongs to the \( EP \) class.

**Proof.** Here we need to show that \( t_p = O(t_1^\epsilon) \) for \( \epsilon < 1 \), where \( t_1 \) and \( t_p \) are the time required using one and \( p \) processors, respectively. From Lemma 4.1, the time required by the algorithm to evaluate the neighborhood by proximity structure is \( O((n^2 \log n) + (\log(n^{\frac{n}{\log n}}))) \). For simplicity lets assume it is \( O(n^2) \). Therefore, it holds that

\[
t_p = O((\text{Neighborhood evaluation time})(\text{neighborhood size}))
= O(n^2 \log n)
= O(n^2 \cdot n^{\delta}) \text{ for } \delta > 0
\]

Now for any \( \delta \) such that \( 0 < \delta < \nu \) and \( \frac{2+\delta}{2+\nu} \leq \epsilon < 1 \) holds \( t_p = O(n^{2+\delta}) = O(n^{(2+\nu)\epsilon}) = O(T_1^\epsilon) \).

Thus, the algorithm belongs to the class \( EP \).

4.3.2 Correctness

In this section, we shall show that the resultant graph produced by the algorithm is a minimum length tree.

**Lemma 4.4.** The tree \( T \) produced by the algorithm is minimal if and only if each \( (u, v) \notin T \) and for all edges \( (x, y) \) in the cyclic graph arising from \( T \) by adding \( (u, v), C_T(u, v) \), we have \( l(u, v) \geq l(x, y) \), where \( l(u, v) \) denotes the Euclidean distance (edge length) between \( u \) and \( v \).

**Proof.** Suppose that the tree \( T \) produced by the algorithm is minimal. If \( l(u, v) < l(x, y) \) then there must exist an edge \( (u, v) \notin T \) and an edge \( (x, y) \in C_T(u, v) \) with \( l(u, v) < l(x, y) \). Since edge \( (x, y) \) is a bridge edge, removal of \( (x, y) \) from \( T \) divides \( T \) into two components. Adding \( (u, v) \) i.e., \((u, v) \cup T - \{x, y\}\) gives a new tree \( T' \). Since \( l(u, v) < l(x, y) \), \( T' \) has smaller length than \( T \). This contradicts the minimality of \( T \).

Conversely if \( l(u, v) \leq l(x, y) \). Suppose algorithm produced a tree \( T \) which spans given set of
Therefore, lemma states that if $T$ of a tree $C$ and $C$ becomes connected again. Therefore, the graph arising from $T$ by adding $(u', v')$, $C_T(u, v')$, has to contain an edge of $T'$, because otherwise $T' \setminus \{u', v'\}$ would still be connected. The minimality of $T'$ implies that $l(u, v) \geq l(u', v')$: replacing edge $(u', v')$ by $(u, v)$ in $T'$, we get another tree $T''$ and if $l(u, v) < l(u', v')$, this tree would have smaller length than $T'$ contradicting the minimality of $T''T$. On the other hand, lemma states that $l(u', v') \geq l(u, v)$, so that $l(u', v') = l(u, v)$ and $l(T'') = w(T')$. Therefore, $T''$ is a minimal spanning tree as well. $T''$ has one more edge in common with $T$ than $T'$, and using induction, we have $l(T) = l(T'') = l(T')$. This completes the proof.

Now we will show the quality of the resultant tree produced by the algorithm by showing that a slight perturbation in terminals (say, $x$, $y$ and $z$) does not affect the final tree produced by our algorithm i.e., does not affect the coordinate of the Steiner point $s$. For this we use the method proposed in [101].

The following lemma is based on [15].

**Lemma 4.5.** Displacement of any terminal along the segment joining the terminal and Steiner point in the neighborhood does not affect the resultant Steiner tree produced by the algorithm.

**Proof.** Consider a neighborhood that consists of segments $\overline{x s}$, $\overline{y s}$ and $\overline{y z}$. Obviously, this is a minimal Steiner tree in the neighborhood of points $x$, $y$ and $z$ with $s$ as a Steiner point. Without loss of generality, suppose there exists a point $z'$ belonging to the line segment $\overline{x s}$. Furthermore, $z' \neq z \neq s$.

We assume that in the SMT produced by the algorithm $s'$ does not coincide with Steiner point $s$ i.e., $s' \neq s$. In addition, the length of this tree consists of the sum of lengths, of segments $\overline{x z'}$, $\overline{s' x}$, and $\overline{s' y}$, is smaller than the total length of the segments $\overline{x s}$, $\overline{s x}$, and $\overline{s y}$, which is also a Steiner tree by definition. In this case, our algorithm reduces the length of the minimum Steiner tree by searching in parallel (note that in this particular case we are working in a single neighborhood) segments $\overline{x z'}$, $\overline{s' x}$ and $\overline{s' y}$ in SEARCH phase and replacing them by segments $\overline{x s'}$, $\overline{s x}$ and $\overline{s y}$ in the Enhancement phase. However, this can not be done because it contradicts the minimality of the tree. \qed
4.4 Execution Time Analysis

This section deals with the time delays among server and client processors for the parallel Steiner tree algorithm presented in [17] and derives the mathematical expressions for the optimal number of processors required by the algorithm in [17]. We start our derivation of the expressions from the point when server has already computed the RNG and MST and is ready to send its solution i.e. server solution to clients. The server processor sends a “server” solution, $\sigma_{\text{server}}$, and subgraph $RNG_i(P_k)$ i.e. partitions, to each of the $C(n)$ clients, using communication time $T_{com}$, where $C(n)$ is the number of clients in server-client system. Note that $T_{com}$ depicts the time to communicate with each client. Next, the server processor evaluates an “enhancement in the previous solution” received from the client using time

$$T_{\text{server}}(n) = \frac{n(n \log n)}{P(n)}$$

where $O(n \log n)$ is the evaluation time required to evaluate the enhanced solution received from the single client, $n$ is the total number of nodes received from all clients, and $P(n) = C(n) + 1$ is the total number of processors used during this course.

In complexity analysis (Section 4.3.1), we assumed that the client processors start working on the solution as soon as they receive the server solution from the server, and return the enhancements in the solution back to server as soon as they finish. This assumption implies that the last client and the server finish the computations at the same time. Asymptotically, there is nothing wrong with this assumption! But in actual experiment there is a time delay $\Delta_{com}$ before the server processor receives the enhancements in the solution and can proceed to evaluate the enhancement. Once we have realized that there is a time delay, the question is whether this time delay is constant or variable. Clearly, the time delay is variable, since it depends not only on the individual communication time between server and clients but also on the client computation. For example, this is certainly possible that a certain client (or number of clients) keeps working on the “solution”, which is completely different from the server’s current solution (global solution). In this case, the server needs to send its current solution and a new partition every time it receives the enhancement from the client. But this requires statistical analysis of the algorithm (program/simulation), which is the goal of our research in progress. Here, in order to construct general expressions, we consider the two common instances. The instance 1 evaluates the computation time in which the communication time $\Delta_{com}$
is constant, while the instance 2 evaluates the computation time in which $\Delta_{com}$ is variable (without specifying the range of variation).

**Case I - When Time Delay $\Delta_{com}$ is Constant**

Allowing all the computational and communication contributions in the equation 4.1, the elapsed time for a single stage (i.e. partition phase, search phase, and merging phase) of the parallel algorithm may be estimated as

$$
T_p = P(n)\Delta_{com} + T_{server}(n)
$$

$$
= P(n)\Delta_{com} + \frac{n(n \log n)}{P(n)}
$$

(4.2)

The second part of equation 4.2 tells us that if we like to decrease the computation time of the algorithm, we must use more client processors. But at the same time, the first part of the equation says that the communication time will increase accordingly. Clearly, these two arguments indicate the tradeoff between computation time and communication time, which in turn implies the existence of an optimal number of processors that minimizes the execution time/communication time. Note that increasing (or decreasing) the number of processor means increasing (or decreasing the number of clients since $P(n) = C(n) + 1$ i.e., the number of server is 1, a constant. To evaluate the optimal number of processors, we differentiate equation 4.2 with respect to $P(n)$ and solve for number of processors, $P(n)$. That is,

$$
\frac{d}{dP} \left[ \frac{(P\Delta_{com} + n(n \log n)}{P} \right] = 0
$$

We get optimal number of processors,

(4.3)

$$
P_{opt}(n) = \sqrt{\frac{(n(n \log n))}{\Delta_{com}}} = n\sqrt{\frac{\log n}{\Delta_{com}}}
$$

Therefore, using the fact that $P(n) = C(n) + 1$, the optimal number of processors i.e., client processors is

(4.4)

$$
C_{opt}(n) = P(n)_{opt} - 1 = n\sqrt{\frac{\log n}{\Delta_{com}}} - 1
$$

Where time delay $\Delta_{com}$ is constant.
Case II - When Time Delay $\Delta_{com}$ is Not Constant

This case is more realistic due to the fact that we are dealing with the Steiner tree problem from computational geometry/graph perspective. Since, the proposed Steiner tree algorithm requires a large amount of data to exchange between processors (server and clients) and since the cost of exchanging communication between two processors depends on the amount of information, $x$. Therefore, we must consider the two important factors in the processors communication namely, network bandwidth and latency of communications, $L$. Recall, the latency, $L$, is the overhead per message that depends on the operating system, the programming environment, and on the particular hardware. Now the server processor sends a $\sigma_{server} = MST(P)$ and $RNG_i(Pk)$ to each of the $C(n)$ clients. According to the above argument, the communication time $T_{com}$ required by the server must include bandwidth of the network and the latency of the communication. If we include the bandwidth and latency in the equation we get

\begin{equation}
T_{com} = Bx + L
\end{equation}

where $B$ is the inverse of the network bandwidth, $L$ is the communication latency, and variable $x$ depicts the amount of information. Even though, each client receives $RNG(Pk)$ of up to 5 nodes [17] (note that this argument originally appeared in [98]), without loss of generality, we assumed that each client receives $n/P(n)$ nodes (terminals). The time to send the node is directly proportional to the inverse of the network bandwidth $B$. If we include the communication latency $L$, the time required to send the $MST(P)$ and $RNG(Pk)$ to the client becomes

\begin{equation}
T_{send} = Bn/P(n) + L
\end{equation}

and the time to receive the enhancement back follows the same logic. That is

\begin{equation}
T_{recv} = Bn/P(n) + L
\end{equation}

Note that for the ease of computation, we ignored the length and fitness value of the server and clients, which requires the statistical analysis. Similar to the Case I, the elapsed time for a single stage (i.e. partition phase, search phase, and merging phase) of the parallel algorithm may be estimated as
Elapsed time = (number of clients)(Time required to send solution to client) + (time required to receive enhanced solution back) + (Server Time to evaluate enhancement).

\[(4.8)\]
\[
T_p = C(n)T_{send} + T_{recv} + \frac{n(n \log n)}{P(n)}
\]

As before, to evaluate the optimal number of processors, we differentiate equation 4.8 with respect to \(P(n)\) and solve for number of processors, \(P(n)\). That is,

\[
\frac{d}{dP} \left[ \frac{C(n)T_{send} + T_{recv} + n(n \log n)}{P} \right] = 0
\]

and solve for \(P(n)\) to obtain

\[(4.9)\]
\[
P_{opt}(n) = \sqrt{n(n \log n) + \frac{B}{L}}
\]

Clearly, the values given by equation 4.3 and equation 4.9, respectively are not very different. The reason is that in most cases the bandwidth is a small number that becomes even smaller as networks get faster. Note that even if we included length and fitness values of the node with the bandwidth \(B\), the resulting number would still be very small. Therefore, we can safely ignore the bandwidth \(B\) in the equation for estimation.

\[
P_{opt}(n) = \sqrt{\frac{n(n \log n)}{L}}
\]

\[(4.10)\]

This is the general expression that gives the optimal number of client processors for which the proposed algorithm in [17] works optimally in terms of running time on the server-client model of computation. The important point regarding this equation is that the dominant factor in the communications time is the latency of the network.

4.5 Simulation

This section presents the event-driven simulation of the parallel Steiner tree.
4.5.1 General Technique

Suppose that there are \( n \) slots in server’s queue and each slot can hold at most one enhanced solution from client with a probability of \( p \) (say). Also, it is reasonable to suppose that enhanced solutions (from same client or different clients) in distinct slots are independent. These assumptions correspond to a Bernoulli process with parameter \( p \). The total number of sub-solutions is then a random variable \( N \) having a binomial distribution with parameters \((n, p)\). The event locations comprise a random binary \( n \)-vector \( V \), in which the first 1’s mark the event locations. However, the chapter uses the direct approach to analyze action (incoming sub-solutions from clients), or lack thereof. We start with a supposition that server admits the subsolutions from clients in a quantized fashion. In fact, this assumption is quite realistic since we are dealing with only one processor i.e. server processor. In particular, the server opens at the beginning of each millisecond to admit a sub-solution with probability \( p = 0.1 \) (say). If the server is busy at that time, the sub-solution enters in a server’s waiting queue. The service policy is first-in, first-out (FIFO) from the queue. Our experiments shows that server takes 4 to 8 milliseconds to responds to a sub-solution either by accepting the sub-solution or rejecting it and sending the new partition to that client.

4.5.2 Halting Condition

Now we would like to show that the simulation algorithm halts when the queue becomes empty. We will constructively establish the halting condition by showing that the server’s waiting queue will not grow indefinitely. Since the service time of the server processor varies in the range of 4 to 8 milliseconds, we say that the service time is a discrete random variable on \( \{4, 5, 6, 7, 8\} \) milliseconds. This implies that for 5 second interval i.e. for 5000 milliseconds, the number of sub-solution received by the server is binomially distributed with probability \( p = 0.1 \) and time interval \( t = 5000 \). Therefore, the server can expect \( t \times p = 5000 \times 0.1 = 500 \) subsolutions on average from clients. This gives us the response time or the service time, which is \((1/2)(4 + 5 + 6 + 7 + 8) = 6.0\) milliseconds. This simply means that the server can handle \( 5000/6 = 833.33 \) subsolutions on average. This calculation constructively showed that the server’s queue will not grow indefinitely.

4.5.3 Simulation Variables

The simulation maintains a list of enhanced solutions from clients in the variable, \( EnhancedSolList \), in the form of vector (state, enterQtime, exitQtime, DepartureSysTime). The first component
of the vector represents the state of the subsolution i.e. 0 and 1 indicate, respectively, the arrival and the departure of the client’s enhanced solution from the waiting server’s queue. The vector component enterQtime represents timestamp in milliseconds for the incoming client’s sub-solutions and exitQtime indicates the timestamp in milliseconds for the outgoing client’s sub-solutions in the waiting server’s queue. The vector components enterQtime and exitQtime represent timestamps for the arrival in and exit from the waiting queue. The last vector component, DepartureQtime indicates the timestamp in milliseconds of the client’s solution, which is far off from the server’s own (global) solution and has to ignore; and update the segment of the solution space assigned to client. That is, DepartureSysTime is the time for a rejected solution from the system.

4.5.4 Overall Strategy

After initialization of the variable EnhancedSolList and scheduling the first event (Lines 1 and 2), the algorithm begins its loop, which repeatedly removes the first enhanced solution from EnhancedSolList for updating server’s solution or global solution. The algorithm halts when the EnhancedSolList becomes empty. Note that we have established the halting condition in section 4.5.2. If server is updating a server solution or attending other clients (serverBusy = 1) and a new Enhanced solution from some client is arrived, it puts the client’s solution in a waitQueue. The two routines enqueue and dequeue insert and remove enhanced solutions from server’s queue, Q. In addition, we need two subroutines to work with the EnhancedSolList; the insert operation puts an enhanced solution in the EnhancedSolList, and remove operation removes the earliest enhanced solution from the EnhancedSolList.

4.6 Algorithm ServerQueue

This section formalizes the arguments of Section 4.5 into following algorithm.
Initialize the EnhancedSolList by setting the vector (state, enterQtime, exitQtime, departureSysTime) (0, 0, 0, 0)

Schedule the first enhanced solution by INSERT(EnhancedSolList, (state, enterQtime, exitQtime, departureSysTime))

while (notEmpty(EnhancedSolList)) do
  Start with earliest solution using FIFO policy
  if State = 0 (incoming solution from clients) then
    Increase number of enhanced solutions by 1
    Mark the arrival time of next subSolution
    Insert the enhanced solution vector (0, j, 0, 0) in EnhancedSolList
    if server is busy in serving other clients then
      ENQUEUE (waitQueue, (State, enterQtime, exitQtime, DepartSysTime))
    else
      Update the server's idle time as
      Total Idle time = total Idle time + enterQtime last service time
      Set State to 1 // change to outgoing solution
      Set exitQtime to enterQtime
      Set departureSysTime to exitQtime
  end
  else if // State = 1 i.e. outgoing solution from queue then
    TotQtime = TotQtime + exitQtime − enterQtime
    TotDepartureTime = TotDepartureTime + DepartureSysTime − enterQtime
    Set CurrentTime to DepartureSysTime
    if (notEmpty(waitQueue)) then
      (state, enterQtime, exitQtime, DepartSysTime) = DEQUEUE(waitQueue)
      Set state to 1 // change to outgoing solution
      Set exitQtime to current time
      Set DepartureSysTime to current time
      INSERT(EnhancedSolList(state, enterQtime, exitQtime, DepartSysTime))
    else
      Set Busy to 0 // set the server free
    end
  end
end

Algorithm 4.3: Algorithm Server (Queue)

4.7 Illustrative Example

Now we scrutinize the algorithm presented in Section 4.6 by following the two vectors in variables EnhancedSolList and waitQueue through a several iterations. During first iteration, the EnhancedSolList contains the first arrival of the enhanced solution from some client at time 1 and an empty waitQueue. The type represents incoming subsolution from client, which is 0. The timestamp of this incoming subsolution (state= 0) is enterQtime = 1 i.e. first subsolution arrived at time 1. Therefore, the vector EnhancedSolList indicates (0, 1, 0, 0). At this point, the variable waitQueue is empty, (0, 0, 0, 0). Upon removing incoming subsolution, the algorithm deals with the next incoming subsolution at time 3 and puts it in the EnhancedSolList, (0, 3, 0, 0). The waitQueue is still
empty, (0, 0, 0, 0). Since, the server is free (serverBusy = 0), the exitQtime becomes the enterQtime. The difference of zero indicates that the enhanced solution from client spends no time in the waitQueue. The algorithm adds the service time, which is 6 millisecond, to the exitQtime to produce the DepartureSysTime, which becomes 7 milliseconds. After changing the state (to state = 1) of client’s enhanced solution, the algorithm places departure enhanced solution in the EnhancedSolList, (0, 3, 0, 0) as we see at the beginning of the second iteration. Table 4.1 provides this information for a short simulation. The values reflect the status at the top of the repeat loop. The second iteration processes the second enhanced solution (from the same client or another), (0, 3, 0, 0) enters the waitQueue, as shown at the beginning of third iteration. The iteration 3 processes the departure enhanced solution. It then removes the waiting enhanced solution from the waitQueue, (0, 3, 0, 0), calculate its departure time, departureSysTime = 13, and places it in the EnhancedSolList. This gives the vector at the beginning of iteration 4. This process continues until server’s queue is empty. While processing a departure of the enhanced solution, the simulation algorithm computes the server’s queue time, which is simply (exitQtime – enterQtime). The algorithm computes the idle time of server while processing the incoming enhanced solution when the server is not busy. Finally, the algorithm outputs the average queue time and total server idle time, depicted in the Table 4.2.

4.8 Summary

In this chapter, we presented a parallel algorithm that utilizes the local search technique for computing a Steiner tree in the two dimensional plane. We implemented the local search technique in parallel for Steiner tree problem that allows us to solve larger problem and improve the quality of the final solution. The main contribution of this work is the $O(n^2 \log^2 n + \log n \log(n^{1/2} n))$ parallel local search algorithm for computing Steiner tree on the Euclidean plane. The main advantage of the algorithm is that it does not need synchronization. As a result, it has no communication overhead. Furthermore, we have presented simulation of the parallel Steiner tree problem on the server-client model of computation. Particularly, we have analyzed the time delays at server end and examined some consequences of time delays on the execution time of the parallel Steiner tree algorithm. In addition, we have also computed the expression for the optimal number of client processors required by the parallel algorithm for optimal efficiency.
The values reflect the status of vectors $\text{EnhancedSolList}$ and $\text{waitQueue}$ at the top of the iteration. Also, IT = Iteration; exitQ = exitQtime; enterQ = enterQtime; ST = State; DST = DepartureSys-Time.

Table 4.1: Vectors Status.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Average Queue Time</th>
<th>Server Idle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12.5</td>
<td>78.6</td>
</tr>
<tr>
<td>2</td>
<td>9.8</td>
<td>165.3</td>
</tr>
<tr>
<td>3</td>
<td>14.4</td>
<td>162.3</td>
</tr>
<tr>
<td>4</td>
<td>9.7</td>
<td>204.3</td>
</tr>
<tr>
<td>5</td>
<td>8.6</td>
<td>338.6</td>
</tr>
</tbody>
</table>

Results of 5 independent simulations in milliseconds.

Table 4.2: Simulation Results.
5.1 Introduction

The concept of Voronoi diagram dates back to 17\textsuperscript{th} century. The mathematicians Dirchlet and Voronoi formally introduced the concept of a Voronoi diagram to mathematics community, and Shamos and Hoey [102] introduced the concept to the field of computational geometry. The concept of Voronoi diagram has been applied in diverse fields of sciences such as anthropology, archeology, astronomy, biology, crystallography, geography, marketing, mathematics, physiology, physics, robotics, statistics, zoology, to name a few. More recently, Voronoi diagram has been used for location updates and routing in wireless networks and mobile computing [103]. For further discussion on applications of Voronoi diagram, see [104]. The Voronoi diagram solves many fundamental proximity problems. Some of these problems include the closest pair problem, which can be stated as find the closest pair of sites among a set of sites, the nearest neighbor problem that is find the nearest neighbor of each of a given set of sites, the convex hull, the Delaunay triangulation and the Euclidean minimum spanning tree. See [104, 105] for details. The idea of this work is derived from the Shamos and Hoey [102]. The published results in the references [18, 22, 24, 66, 106] are the basis of the chapter.

5.1.1 Our Contribution

The main contribution of this chapter is an $O(lg^3n)$ time parallel algorithm for constructing the Voronoi Diagram of a set of $n$ sites in the Euclidean plane on a hypercube with $n$ processors. Our technique parallelizes the well-known seemingly inherent sequential technique of Shamos and Hoey, and makes use of a number of special properties of the dividing polygonal chain and the Batcher’s bitonic sort. Our approach is based on the divided chain technique presented in [107], the proposed algorithm finds the sites by first computing intersection points in parallel between bisectors and Voronoi edges that are used in the construction of the dividing chain. Reader may consult [66].
5.2 The Overall Strategy of An Algorithm

First sort the \( n \) planar sites belonging to the given set \( S \) in parallel according to abscissa using \( O(n) \) PEs of hypercube in \( O(lg^2 n) \) time using bitonic sort [108]. Then partition the \( n \) sites such that each partition gets at most one site from original set \( S \). Now map each subset or equivalently each site onto PEs such that \( PE_0 \) gets the site with the lowest \( x \)-coordinate and \( PE_{n-1} \) gets the site with the highest \( x \)-coordinate. In the merging phase all PEs compute \( n/2 \) sub-Voronoi diagrams. Note that in this case they are just \( n/2 \) bisectors. \( PE_i \) and \( PE_{i+1} \) compute the perpendicular bisector of sites \( s_i \) and \( s_{i+1} \) in \( S \) where \( i = 0, 2, 4, \ldots, n-2 \). We name the bisectors \( B(s_i, s_{i+1}) \), the \( \text{Vor}_1(i) \) and the bisector \( B(s_{i+2}, s_{i+3}) \), the \( \text{Vor}_1(i+1) \) where \( i = 0, 2, 4, \ldots, n-4 \). In the second stage, we merge pairs of \( \text{Vor}_1(i) \) and \( \text{Vor}_1(i+1) \) to form \( \text{Vor}_2(i) \) and \( \text{Vor}_2(i+1) \). This gives the resulting sub Voronoi diagrams. In this manner, we recursively stitch the sub-Voronoi diagrams \( \text{Vor}_d(i) \) and \( \text{Vor}_d(i+1) \) for \( 2 \leq d \leq lg(n) \). In the last stage of merging, we stitch sub-Voronoi diagrams \( \text{Vor}_{lg(n)-1}(i) \) and \( \text{Vor}_{lg(n)-1}(i+1) \) to shape a complete (inclusive) Voronoi diagram, \( \text{Vor}(S) \).

Like this sequential counterpart [102] of this algorithm, the most vital part of stitching progression is the creation of the dividing chain. Our algorithm creates the dividing chain by first searching the intersection sites between bisectors and Voronoi edges. Next, we outline the course of action to build this intersection set.
5.3 Description of Merging

In each merging step, \(2 \leq d \leq \lg(n)\), define all unbounded sites whose associated Voronoi polygons are infinite in partition sets. We call such sites unbounded. Define sites that have maximum \(y\)-coordinates and minimum \(y\)-coordinates in subsets \(S_i\) (subset strictly on the left) and \(S_{i+1}\) (subset strictly on the right) as \(ymax\) and \(ymin\) respectively. The unbounded sites arranged in the counterclockwise direction from \(ymax\) to \(ymin\) are identified as \(\alpha_i\) sites where \(ymax \leq i \leq ymin\) and those sites that are arranged in clockwise direction from \(ymax\) to \(ymin\) be defined as \(\beta_i\) sites, where \(ymax \leq i \leq ymin\).

In subsequent merging stage, every \(PEs\) that holds \(\alpha\) sites in subset \(S_i\) will search in parallel for \(\beta\) sites in the subset \(S_{i+1}\), which has the smallest \(y\)-coordinate greater than the \(y\)-coordinate of its own. After this search, \(PEs\) of \(\alpha\) sites will search for \(\beta\) sites in the strictly right subset \(S_{i+1}\), which has largest \(y\)-coordinate that is smaller than the \(y\)-coordinate of its own. The search of these sites can easily be accomplished using binary search on unbounded sites in \(O(lg(n))\) time. Quinn [109] discussed in detail the parallel search and embedding various graphs into hypercube.

After the detection of sites, we compute the perpendicular bisector, \(B(\alpha_i, \beta_{i+1})\), of unbounded sites in the strictly left partition, \(\alpha_i\), and unbounded sites in strictly right partition, \(\beta_{i+1}\). Then find the intersection site between Voronoi edges of sites \(\alpha_i\), and \(\beta_{i+1}\). We can accomplish this task by using “walking method” described in incremental construction of Voronoi diagram [110,111].

After the detection of intersection site between Voronoi edge and perpendicular bisector, \(B(\alpha_i, \beta_{i+1})\), include it in the set of intersection, \(X\). Simultaneously, algorithm runs this procedure for all \(\alpha\) and \(\beta\) sites in strictly left and strictly right partitions respectively. The consequence, after running this procedure on all \(lg(n)\) dimensions of \(O(n)\) hypercube, is the finished Voronoi diagram of the given set of sites, \(S\).

5.4 The Algorithm

This section presents the parallel algorithm for Voronoi diagram construction using the hypercube model of computation.
Input : A set of $n = 2^d$ planar sites
Output: A set of Voronoi edges specified by set of Voronoi edges

1. Sort the $n \in S$ sites by their $x$-coordinates using bitonic sort and map $n$ sites $s_0, s_1, \ldots, s_{n-1}$ onto the $O(n)$ PEs such that each $PE_i$ obtain site $s_i$, $0 \leq i \leq n - 1$.

2. for all $0 \leq i \leq n - 1$ in Parallel do
   3. for all $0 \leq i \leq n - 1$ do
      4. Construct perpendicular bisector $B(s_i, s_{i+1})$ using $PE_i$ and $PE_{i+1}$ for $i = 0, 2, 4, \ldots, n - 2$.
   5. end
   6. for all $2 \leq k \leq \log(n)$ do
      7. Merge $Vor_{d-1}(S) = Vor_{d-1}(i) \cup Vor_{d-1}(i + 1)$ by constructing the set of Intersection, $X$.
   8. end
9. end
10. Sort intersection points in $X$, with respect to their $y$-coordinates using bitonic sort.
11. Add polygonal line and discard the portion of $Vor_{d-1}(i)$ which is strictly left of the polygonal line and discard the portion of $Vor_{d-1}(i + 1)$ which is strictly right of the polygonal line.

Algorithm 5.1: Construction of Voronoi Diagram on Hypercube

Following algorithm computes the intersection points between bisectors and Voronoi edges that are used in the construction of dividing polygonal chain.

1. Search unbounded sites in strictly left subset, $S_L$, and strictly right subset, $S_R$.
2. Search for the maximum and minimum $y$-coordinates sites in subsets $S_L$ and $S_R$ such that $ymax, ymin \in S_L$ and $ymax, ymin \in S_R$.
3. Identify $\alpha$ and $\beta$ sites $S_L$ and $S_R$ respectively where, $ymax \leq i \leq ymin$.
4. In parallel, every PE that hold $\alpha \in S_L$ will search for $\beta \in S_R$, which has smallest $y$-coordinate greater than its own. Simultaneously, every PE of $\beta \in S_R$ will search for $\alpha \in S_L$, which has largest $y$-coordinate that is smaller than its own.
5. In parallel, do step 2 after interchanging th role of $\alpha$ and $\beta$ sites.
6. Compute the perpendicular bisector of $\alpha \in S_L$ and $\beta \in S_R$, $S = S_L \cup S_R$. That is, $B(\alpha_L, \beta_R)$.
7. Search the intersection points between Voronoi edges associated with Voronoi polygons of $\alpha \in S_L$ and $\beta \in S_R$ sites perpendicular bisector $B(\alpha, \beta)$.
8. Include intersection points found in step 5 in the intersection set, $X$, i.e., $X \leftarrow X \cup \{x\}$
9. Return intersection set, $X$.

Algorithm 5.2: Construction of Dividing Polygonal Chain

5.5 Construction of the Intersection Set

Now, we scrutinize the algorithm by merging two Voronoi diagrams. Suppose that we have obtained the left Voronoi diagram $Vor_L$ and the right Voronoi diagram $Vor_R$ of sets $S_L$ and $S_R$ where $S = S_L \cup S_R$. 
Search all unbounded sites in strictly left and strictly right partitions $S_L$ and $S_R$ respectively. Consider only those sites in $S_L$ that are arranged in counterclockwise order from highest $y$-value to lowest $y$-value. Similarly, consider only those sites in $S_R$ that are arranged in clockwise order from highest $y$-value to lowest $y$-value. We have $\{6, 7, 3\} \in S_L$ and $\{13, 9, 8, 10\} \in S_R$.

First, we construct the semi-infinite ray by computing perpendicular bisector, $B(6, 13)$, of sites 6 and 13. Note that these sites have maximum $y$-coordinates in their respective sets. Then find the intersection point of bisector $B(6, 13)$ and $V(13)$. Let it be point $x_1$. Similarly, construct semi-infinite ray using sites 3 $\in S_L$ and 10 $\in S_R$; and find the intersection point $x_6$.

At this stage, we have set of intersection $X_1 = \{x_1, x_6\}$. Now site 9 $\in S_R$ has largest $y$-coordinate smaller than $y$-coordinate of 6 $\in S_L$. Compute the bisector $B(6, 9)$ and find the intersection points of bisector $B(6, 9)$ and $V(9)$, which are $x_1$ and $x_2$. Include these intersection points in the set of intersections and we get $X_1 = \{x_1, x_2, x_6\}$. The next unbounded site in the set $S_L$ is site 7. Site 8 $\in S_R$ has the smallest $y$-value larger than 7 $\in S_L$. Again, compute the bisector $B(7, 8)$, and find the intersection point of $B(7, 8)$ and Voronoi edges of site 7 and 8, which are $x_4$ and $x_5$. Include these points in the set of intersection, which now becomes $X_1 = \{x_1, x_2, x_4, x_5, x_6\}$.

Simultaneously, processors of set $S_L$ run the same procedure and get the set of intersection points, $X_2 = \{x_1, x_2, x_3, x_5, x_6\}$. The union of sets $X_1$ and $X_2$ gives a set of intersection $X = X_1 \cup X_2 = \{x_1, x_2, x_3, x_4, x_5, x_6\}$. Now, sort the set of intersection with respect of $y$-coordinate using bitonic sort and add the polygonal line $(x_1x_2, x_2x_3, x_4x_5, x_5x_6)$. Finally, delete from $Vor_L$ the part to the right of the polygonal line and delete from $Vor_R$ the part to the left of the polygonal line and return the resultant Voronoi diagram.

5.6 Algorithm Complexity

**Theorem 5.1.** The Voronoi diagram of a set of $n$ sites in the plane can be constructed on a $O(n)$ hypercube computer with in $O(\log^3(n))$ time.

**Proof.** In the sorting phase of an algorithm, we can sort $n$ planar points on hypercube using bitonic sort in $O(\log^3(n))$ time. The partition phase i.e., distribution of $n$ planar sites on $O(n)$-hypercube can be done in constant time since all geometric operations at this stage take constant amount of time. Note that this argument is based on the fact that the planar sites have already been sorted. It is easy to see that the conquer phase can be performed in constant time. In the merging phase, each $PE$ builds a new Voronoi diagram by calling several sub algorithms for finding the perpendicular
bisector, deleting Voronoi edges, new Voronoi edges, etc. Each of these sub algorithms takes a constant time. Suppose, the running times of these sub algorithms be $c_1, c_2, \ldots, c_n$ then we have $C = \max(c_1, c_2, \ldots, c_n)$. The searching in this phase for sites associated with infinite polygons take $O(\lg(n))$ time using binary search. Hence, running time of merging step is no more than $O(\lg(n))$, since $C$ is a constant. Moreover, sorting the set of intersection with respect to $y$-coordinate using bitonic sort takes $O(\lg^2(n))$ time and adding polygonal line takes constant time.

Now, let $t(n)$ denote the time required by the algorithm for $n$ sites. Then, we get the recurrence relation $t(n) = t(n/2) + O(\lg^2(n))$. This equation implies that $t(n) = O(\lg^3(n))$. Note that communication overhead is not included in the solution. Since in all merging stages, messages are exchanged between PEs that are exactly one hop away therefore, the lengths of communication paths are always one. Clearly, sending and receiving a message from one node (PE) to other node (PE) of hypercube takes a constant time. Suppose, in $d^{th}$ dimension, communication required time $C_d$. Hence, the total time required by all dimensions, ($\lg(n)$), of hypercube would be $t_{\text{com}} = C_{\text{com}}(n/2) + C_d = C_{\text{com}}(n/2) + C_d(\lg(n)) \leq 2 \times C \times \lg(n) = \lg(n)$. Hence, the total communication time to build a complete Voronoi diagram would be $t_{\text{com}} = O(\lg(n))$. From these two pieces of information, we get $t(n) = t(n) + t_{\text{com}}(n) = O(\lg^3(n)) + O(\lg(n)) = O(\lg^3(n))$. Therefore, the Voronoi diagram of a set of $n$ sites in the plane can be computed on an $O(n)$-hypercube computer in $O(\lg^3(n))$ time. This completes the proof.

5.7 Correctness

In this section, we prove the correctness of an algorithm by showing that the algorithm will result in the Voronoi diagram of the given set of planar sites. But before stating and proving lemmas and theorem, we outline the procedure. We need to prove that this algorithm will contain all Voronoi vertices and edges contained in $\text{Vor}(S)$, and that every vertex and edge that is retained by the algorithm is contained in $\text{Vor}(S)$. In this algorithm, the individual edges of the divided chain can be obtained by joining intersection points in the set $X$. Since edges of the divided chain are true Voronoi edges, we can derive their properties by exploiting the established properties of the Voronoi diagram [67]. In this correctness, we demonstrate that the edges of polygonal line are indeed edges of divided chain and further the algorithm constructs every edge of divided chain by using intersection set $X$. Following lemmas based on [67].

**Lemma 5.1.** Only the closest sites belong to other partition participated in the constructing of
Proof. Consider a site $s_j$ be the closest to $s_i$ and let $v$ be the midpoint of segment $s_is_j$. Now, suppose the $v$ does not lie on the part of dividing chain corresponding to $s_i$ and $s_j$. Then the line $s_iv$ intersects some part of dividing chain corresponding to $s_i$ and $s_k$ (say) at point $u$. Which means, the length $|s_iv| < |s_is_j|$, therefore length $|s_iv| < 2|s_is_k|$. But since $2 \times |s_is_v| = |s_is_j|$, implies that the length $|s_is_k| < |s_is_j|$. And we would have $s_k$ closer to $s_i$ than $s_j$, which is a contradiction.

**Lemma 5.2.** The site(s) participated in the construction of dividing chain if and only if it is unbounded.

Proof. Consider sites $s_1, s_2, s_3, s_i \in P_L$ such that $s_i$ is bounded and a site $x \in P_R$. In the partition $P_L$, there must exist three Voronoi circles $C_{12}, C_{13}$ and $C_{23}$ containing the common site $s_i$. That is, $C_{12} = \{s_i, s_1, s_2\}, C_{23} = \{s_i, s_2, s_3\}$ and $C_{13} = \{s_i, s_1, s_3\}$. WLOG, consider circle $C_{12}$ with vertices $\{s_i, s_1, s_2\}$ on the boundary of circle. Also consider an external arc $A_{12}$ on the circle $C_{12}$ between sites $s_1$ and $s_2$ not containing $s_i$. We claim that any site $x \in P_R$ is closer to one of the sites $s_1, s_2$ or $s_3$ than it is to $s_i$. Consider the segment $xs_i$. By the Jordan curve theorem, segment $xs_i$ must intersect one of the sides of triangle $s_1s_2s_3$. WLOG, consider it intersects $xs_i$; therefore, segment $xs_i$ also intersects arc $A_{12}$ at point $y$. But $y$ is closer to either $s_1$ or $s_2$ than to $s_i$, hence the claim. Since point $x \in P_R$ is closer to $s_1, s_2$ or $s_3 \in P_L$ than to $s_i \in P_L$, only $s_1, s_2$ or $s_3$ of partition $P_L$ participated in construction of dividing chain with site $x \in P_R$.

Conversely assume that site $s_i$ is bounded and let the sequence of boundary edges $e_1, e_2, \ldots, e_k$ where $k \geq 3$. Each boundary edge belongs to the part of bisector. Hence the lemma.

**Lemma 5.3.** The unbounded site that is closest to some unbounded site in other partition involves in construction of dividing chain.

Proof. Immediately from lemma 5.1 and lemma 5.2.

Now we show that the algorithm can construct the part of the dividing chain associated with bounded site, $s_k \in P_L$. By lemma 5.2, site $s_y \in P_R$ constructs the bisector (part of dividing chain) with $s_i \in P_L$ or $s_j \in P_L$ (not with $s_k \in P_L$ since it is bounded). From the intersection of bisector and Voronoi edge, we get the intersection point. WLOG assume it is $x_1$. After constructing a bisector with site $s_i$ or site $s_j$ (whichever be the closest, see lemma 5.1), the remaining site becomes closest.
Therefore, site \( s_y \in P_R \) constructed the bisector with that site and so we have intersecting point \( x_2 \) on Voronoi edge.

Since, \( x_1 \) is a center of circumcircle corresponding to \( s_k, s_y \) and \( s_i \); and \( x_2 \) is a center of circum-circle corresponding to \( s_k, s_y \) and \( s_j \) (Note that \( s_k \) and \( s_y \) are common in both circles). We simply join the intersection points \( x_1 \) and \( x_2 \) to get a Voronoi edge associated with site \( s_k \) and site \( s_y \), i.e., bisector \( B(s_k, s_y) \), hence the part of divided chain.

5.8 Summary

The chapter addresses the problem of constructing the planar Voronoi diagram in parallel on the hypercube model of computation. While the more novel results have been in the area of parallel algorithms, real focus of this chapter is to combine theoretical and practical results to develop a practical parallel algorithm by exploiting the characteristic properties of computational model. In this chapter, we have shown that the problem of constructing planar Voronoi diagram can be solved using hypercube model of computation with \( n \) processors in \( O(\log^3(n)) \) time. The fact that the bitonic sort algorithm bounds the time complexity of the algorithm also showed that sorting helps in the constructing of Voronoi diagram.

The fundamental problem in the constructions of sequential Voronoi diagram is the construction of polygonal line quickly. This chapter shows that this fundamental problem remains unchanged even in the parallel setting. One of the features of the algorithm presented is to construct a polygonal dividing chain from intersection points. Hence, we have shown that it is certainly possible to construct the polygonal chain without explicitly computing all part of it.
6.1 Introduction

An ad hoc wireless network is a collection of independent devices (transceiver nodes) that have to communicate themselves in the absence of any central authority. The absence of central authority implies that the coordination necessary for communication has to be carried out by the nodes themselves. Ad hoc network nodes support direct communication using wireless transceivers without need for a fixed infrastructure. Any two devices (nodes) in the region achieve communication either directly if they are within each others transmission range (i.e., within one hop) or indirectly via other nodes, if they are out of each others range (multi-hop). Note that in this work we assume that transceiver nodes are portable devices. Typically, portable devices have limited power resources available. One of the main characteristic properties of the ad hoc wireless networks is the ability to vary the power used in communication among nodes, which implies the concept of transmission range, hence the range assignment. The published results in the references [2,4,6,8,13,14,16,19,21] are the basis of the chapter.

A range assignment for the finite set $N$ of $n$ network nodes is a function, $RA : N \rightarrow R^+$ (where $R^+$ is the set of non-negative numbers), that assigns to every node $u \in N$ such that $0 < RA(u) \leq r_{\text{max}}$, where $r_{\text{max}}$ is the maximum transmitting range. The cost of a range Assignment is defined as $\text{cost}(RA) = \Sigma (RA(u))^\alpha$ for some real constant $\alpha > 0$. The underlying intuition is that the element of a finite set $N$ are given transceiver nodes, and one can choose for each transceiver nodes $u \in N$ a corresponding data transmission range i.e., radius, $RA(u)$. Sending data at radius, $RA(u)$, consumes energy proportional to $(RA(u))^\alpha$. The parameter $\alpha$ is called the distance-power gradient. In an ideal environment, i.e., in the empty space $\alpha = 2$ but it can vary from 1 to more than 6 depending on the environmental conditions of the place where the transceiver nodes are located, see [112].

Let $\text{dist}(u, v)$ denote the Euclidean distance between two arbitrary nodes $u, v \in R^2$. We define
range assignment $RA$ on the following kind of graph. Let $G_{RA} = (N, E_{RA})$ be the directed communication graph of range assignment $RA$ where an edge (communication link) belongs to the set $E_{RA}$ if, and only if, node $u$ can send the data to node $u$, i.e. the radius of node $u$ is at least as large as the distance between nodes $u$ and $v$. Formally, we say as follows:

**Proposition 1.** Given nodes $u, v \in (N, E_{RA})$, edge $(u, v) \leq E_{RA}$ if, and only if, $RA(u) \geq \text{dist}(u, v)$.

![Example of two-dimensional communication graph. Note that three links in the graph are unidirectional.](image)

Figure 6.1: Example of two-dimensional communication graph. Note that three links in the graph are unidirectional.

In this work, we demand that the induced communication graph $G_{RA}$ must be strongly connected to ensure all-to-all communication. Note that here by strong connectivity we mean the relevant optimization problem, i.e. the problem to find a range assignment $RA$ that has minimal $cost(RA)$ among all range assignments satisfying the strong connectivity property.

**Proposition 2.** A given directed communication graph $G_{RA}$ is strongly connected if every two node $u, v \in G_{RA}$ can send data to each other i.e. $u$ and $v$ are reachable from each other.

6.1.1 Our Contribution

The chapter describes an efficient method for introducing relay nodes in the given communication graph. Our algorithm assigns transmitting ranges to the nodes such that the cost of range assignment function is minimal over all connecting range assignments in the graph. The main contribution of the chapter is the $O(N \log N)$ algorithm to add relay nodes to the wireless communication network and 2-approximation to assign transmitting ranges to nodes (original and relay). It does not assume that communication graph to be a unit disk graph. The output of the algorithm is the minimal Steiner tree on the graph consisting of terminal (original) nodes and relay (additional) nodes. The
output of approximation is the range assignments to the nodes.

6.1.2 Chapter Guide

Section 6.2 formulates the problem of range assignment in the geometric graph. The formulation includes the addition of Steiner points so that the graph is strongly connected and the cost of a range assignment function is minimal over all connecting range assignment functions.

Section 6.3 presents the overall strategy of the proposed algorithm for the range assignment problem. We formalize the idea of the proposed algorithm and establish the correctness.

Section 6.4 presents an algorithm that 2-approximates the solution of the range assignment problem of the preceding section. In addition, the section gives a brief introduction to the properties of the proposed scheme.

Section 6.5 discusses the open question related to the proposed scheme.

Section 6.6 contains the summary of the chapter.

6.2 Problem Formulation

We shall formulate the problem in the geometric graph, using [113] and [16], as follows. Given a set \( N \) of \( n \) points, the problem is to introduce a minimal number of additional points such that the graph \( G(N \cup N_{RA}, RA) \) is strongly connected and the cost of the range assignment function, \( \text{Cost}(RA) = \sum (RA(u))^\alpha \) is minimum over all connecting range assignment functions, where \( \alpha \) is the distance-power gradient. Note that if range assignment function \( RA = 0 \) (i.e. transmitting range \( r = 0 \)), this problem reduces to finding the Euclidean Steiner tree problem [17] for the set \( N \) and in context of ad hoc wireless networking, the solution of this problem is proposed in [16] and [15].

Given the location of the terminal nodes, we are interested in the problem of establishing strong connectivity by minimizing the number points (physical locations) where the support team should placed additional relay nodes and minimizing the range assignment, \( RA \), of the resultant graph (tree). In a certain sense, this problem can be seen as a generalization of the problem of determining the critical transmitting range for connectivity, where the constraint that all the nodes have the same transmitting range is dropped. Note that the solution of the problem in which given nodes have the same transmitting range is proposed in [13] and [113]. Formally, let \( N \) be a set of terminal nodes, with \( |N| = n \). These nodes are located in a certain bounded convex region \( S \) of the Euclidean space \( R^2 \). The problem instance is completely defined by the set of locations of \( n \) terminal nodes, \( N = \{ x_i \in S : 1 \leq i \leq n \} \), and the range assignment function \( RA \). The resultant network topology
will be a directed graph $G(N, E(N, RA)) = G(N, RA)$, where $E(N, RA)$ is an edge set defined by $E(N, RA) = \{(x_i, x_j) : x_i, x_j \in N, i \neq j, ||x_i - x_j|| \leq RA\}$. A solution to the problem is a set of locations of relay nodes, $N_{RA} = \{y_i \in S : 1 \leq i \leq N_{RA}\}$ and range assignment function $RA$ and represented in a single maximal connected component in the graph $G(N \cup N_{RA}, RA)$.

6.3 Overall Strategy of the Algorithm

For the ease of the presentation, we divide the overall algorithm’s strategy into four distinct phases namely, construction phase, test phase, merge phase, and the reduction phase. The preliminary version (Kruskal’s version) of this work appeared in the Reference [6]. Our algorithm operates much like Prim’s algorithm for finding the minimum spanning tree. This algorithm has the property that the adding of edges in the set always forms a single tree [67]. As illustrated in the example below, the algorithm starts by placing terminal in the priority queue. The tree starts from an arbitrary terminal and grows from there until the tree spans all the vertices in $N$. At every stage, it selects the shortest available edge that can extend the tree to an additional terminal and marks the terminals up to 4-terminals for which edges are making right turns or left turns. The algorithm ends the construction phase by partitioning the set into subsets of 4-, 3- and 2-terminals using markings. Note that since we are using right turn (left turn) technique to create a subset, at least one terminal is common in adjacent subsets. The performance of this phase depends on how we implement the priority queue. If queue is implemented as a binary heap, building heap requires $O(n)$ time. Since each operation that requires to extract the minimum runs $O(\log n)$ time, and there are $O(n)$ such operations therefore, the total time is $O(n \log n)$. Selecting the shortest edge from the neighboring edges requires $O(n)$ time. Thus, the total time for construction phase is $O(n \log n)$. However, the asymptotic running time of this phase can be improved by implementing priority differently e.g. Fibonacci heap.

In the test phase, the algorithm first performs the convexity test using Lemmas 2.1 and 2.2 for 3-terminal subset and 4-terminal subset. After that, it performs the length test to find out if there is any edge in the subsets whose length is greater than $\delta > 0$. If such an edge exists in 3-terminal and 4-terminal subsets, additional points (relay sensors) are required to increase the lifetime of network. Subsets that pass the convexity test are those whose overall lengths can be minimized by adding Steiner points. Hence, increase the lifetime of network. The algorithm includes the additional point in 3-terminal subsets using the algorithm in [114] (or see Figure 2.3) and additional two points in
4-terminal subsets using simple geometry in [115] (or see Lemma 2.2 or Figure 2.5). It is easy to see that operations in this phase require constant time. Therefore, this phase require linear time.

In the merging phase, the algorithm simply combines the 2-terminal, 3-terminal and 4-terminal subset by union operation. Since at least one terminal is common in adjacent subsets, union operation will not create the cycles.

Lastly, the reduction phase computes the lengths of each edge in the tree produced by merging phase in $O(n)$ time. Sort the edges in non-decreasing order of lengths using some optimal sorting algorithm (e.g. heap sort) that require $O(n \log n)$ time. For all edges, if the length of edge is greater than $\delta$ then add $\left(\left\lceil \frac{|e_i e_{i+1}|}{\delta} \right\rceil - 1 \right)$ 2-degree points which can be done in $O(n)$ time. Thus, the total time for reduction phase is $O(n \log n)$.

The output of the algorithm is the tree that spans all the terminals of the given set $N$ whose each edge has a length at most given positive $\delta$.

6.3.1 Algorithm ADD_NODE

In this section, we formalize the idea presented in the preceding section. The goal of our algorithm is to abandon long-distance edges (communication links) and establish small edges instead. For this purpose, the algorithm chooses a location of additional nodes or relay node based on the Steiner topology. The following algorithm is based on the Prim’s algorithm appeared in [16] in the context of energy conservation in wireless networks and the version based on the Kruskal’s algorithm appeared in [6].

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Given a set $N$ of $n$ terminals. Place $n_i \in N$ in a priority queue. Call this priority queue $Q$</td>
</tr>
<tr>
<td>2</td>
<td>For each $n_i \in Q$, select the shortest available edge in $\text{Adj}[n_i]$</td>
</tr>
<tr>
<td>3</td>
<td>Check whether new edge making right turn (left turn) with previous edge</td>
</tr>
<tr>
<td>4</td>
<td>Mark all terminals (up to 4 terminals) corresponding to consecutive edges that are making right turns (left turns). At the end of this step, we will have connected sub-graphs consisting of 4-, 3-, and 2-terminals</td>
</tr>
<tr>
<td>5</td>
<td>Check the convexity of the 3- and 4-terminal subsets using Lemmas 2.1 and 2.2 respectively and include additional points if sub-graphs pass these tests</td>
</tr>
<tr>
<td>6</td>
<td>Merge 2-, 3-, and 4-terminal subsets (sub-graphs)</td>
</tr>
<tr>
<td>7</td>
<td>Suppose $e_1, e_2, \ldots, e_{n-1}$ are all edges in the resultant tree. Compute $l(e_i)$ for each edge $e_i$</td>
</tr>
<tr>
<td>8</td>
<td>Sort the edges $e_i$, $1 \leq i \leq n - 1$, in a non-decreasing order of $l(.)$</td>
</tr>
<tr>
<td>9</td>
<td>For all $n-1$ edges, If $</td>
</tr>
</tbody>
</table>

Algorithm 6.1: ADD_NODE
6.3.2 Correctness

In this section, we shall show that the resultant graph produced by the algorithm is a minimum length tree. Following lemma is based on [13] and [16].

**Lemma 6.1.** The tree $T$ produced by the algorithm is minimal if and only if each $(u,v) \notin T$ and for all edges $(x,y)$ in the cyclic graph arising from $T$ by adding $(u,v), C_T(u,v)$, we have $l(u,v) \geq l(x,y)$, where $l(u,v)$ denotes the Euclidean distance (edge length) between $u$ and $v$.

**Proof.** Suppose that the tree $T$ produced by the algorithm is minimal. If $l(u,v) < l(x,y)$ (i.e., condition given in the Lemma 2.1 does not satisfy) then there must exist an edge $(u,v) \notin T$ and an edge $(x,y) \in C_T(u,v)$ with $l(u,v) < l(x,y)$. Since edge $(x,y)$ is a bridge edge, removal of $(x,y)$ from $T$ divides $T$ into two components. Adding $(u,v)$ i.e., $(u,v) \cup T - \{x,y\}$ gives a new tree $T'$. Since $l(u,v) < l(x,y)$, $T'$ has smaller length than $T$. This contradicts the minimality of $T$.

Conversely if $l(u,v) \leq l(x,y)$. Suppose algorithm produced a tree $T$ which spans given set of points and show that some minimal spanning tree $T'$ produced by same set of points is same as that of $T$ in terms of length i.e., $l(T) = l(T')$. This will prove that $T$ is minimal as well. We use induction on the edge $m \in T' - T$. When $m = 0$, $T = T'$ is trivial. So let $(u',v') \in T' - T$. If we remove $(u',v')$ from $T'$, the $T'$ is divided in two connected components $C_1$ and $C_2$. If we add the path $C_T(u',v') - \{u',v'\}$ to $T' - \{u',v'\}$ i.e., $T' - \{u',v'\} \cup C_T(u',v') = \{u',v'\}$, connected components $C_1$ and $C_2$ becomes connected again. Therefore, the graph arising from $T$ by adding $(u',v')$, $C_T(u,v')$, has to contain an edge of $T'$, because otherwise $T' - \{u',v'\}$ would still be connected. The minimality of $T'$ implies that $l(u,v) \geq l(u',v')$: replacing edge $(u',v')$ by $(u,v)$ in $T'$, we get another tree $T''$ and if $l(u,v) < l(u',v')$, this tree would have smaller length than $T'$ contradicting the minimality of $T'$. On the other hand, lemma states that $l(u',v') \geq l(u,v)$, so that $l(u',v') = l(u,v)$ and $l(T'') = l(T')$. Therefore, $T''$ is a minimal spanning tree as well. $T''$ has one more edge in common with $T$ than $T'$, and using induction, we have $l(T) = l(T'') = l(T')$. This completes the proof. \(\square\)

6.4 Approximations for Range Assignment

In this section, we present an approximation of the optimal solution of range assignment problem by using the algorithm ADD\_NODE. In essence, we are defining the range assignment, RA, on the tree produced by the ADD\_NODE algorithm.
Algorithm 6.2: Range Assignment Approximation

Now we show that the above “approximation algorithm” produces a 2-approximation of the optimal solution. The following lemma is based on [116].

**Lemma 6.2.** Let $RA_{opt}$ be an optimal range assignment of the problem and $RA_T$ be the range assignment defined above [see line 2 of the Approximation]. Then $\text{Cost}(RA_T) < 2 \times \text{Cost}(RA_{opt})$.

**Proof.** For the proof, we use the technique described in [77]. It consists of two steps. First, we show that $\text{Cost}(RA_{opt})$ is greater than the cost of tree produced by the algorithm i.e., $\text{Cost}(T)$. Then, we show that cost of range assignment on the resultant tree, $\text{Cost}(RA_T)$ is less than twice of the cost of resultant tree i.e., $\text{Cost}(RA_T) < 2 \times \text{Cost}(T)$. Firstly, $\text{Cost}(RA_{opt}) > \text{Cost}(T)$: For optimal assignment a given set $N$ of $n$ points, construct a shortest path destination tree rooted at $u$. Given the shortest path destination tree, the correspondent tree $T'$ (say) can change the directed edges. Since each of the $n - 1$ nodes other than the root must be assigned a range, we have $\text{Cost}(RA_{opt}) > \text{Cost}(T')$. Secondly, since each edge of the resultant tree $T$ can be chosen at most by two nodes, we have $\text{Cost}(RA_{opt}) < 2 \times \text{Cost}(T)$.\hfill \Box

Now we show that the Approximation algorithm satisfies the strong connectivity property (Proposition 2) i.e. the resultant graph (tree) produced by the Approximation is strongly connected.

For this purpose, we define the minimal distance between $u$ and $v$ over all nodes as follows: $dist_u(N) = \min\{\text{dist}(u,v) : u \in N \text{ and } v \in N \setminus \{u\}\}$. Note that this idea depends upon the notion of well-spread defined in [117]. We claim that for a given set of nodes $N$, a range assignment $RA$ in the approximation, the condition $RA(u) \geq dist_u(N)$ must hold for all nodes in the set $N$. We formalize our claim as follows:

**Lemma 6.3.** For a given set of nodes $N$, if a range assignment $RA$ allocated by the Approximation is strongly connected, then $RA(u) \geq dist_u(N)$ for all $u \in N$.

**Proof.** Assume to the contrary, that there exists a range assignment $RA$ in the Approximation such that $RA(u) < dist_u(N)$. That is, condition given in line 2 of the Approximation algorithm does not hold for some particular but arbitrarily chosen nodes $u$ and $v$. Then, by the Proposition 1, if there
was a node \( u \) with \( RA(u) < dist_u(N) \), it could not send data to any other node (terminal or relay node). This implies that there cannot be an edge (communication link) leaving the node \( u \). Thus, \( G_{RA} \) cannot be connected. Since \( G_{RA} \) is not connected therefore, it cannot be strongly connected. This contradicts the supposition that \( RA(u) < dist_u(N) \). Hence, the supposition is false and the lemma is true. This completes the proof.

Indeed, the same node connectivity adopted in the Lemma 6.3 can be used to prove the following stronger result about the connectivity in the resultant graph produced by the algorithm ADD_NODE. In particular, the following lemma shows that \( G_{RA} \) preserves the connectivity of \( G_{ADD\_NODE} \).

**Lemma 6.4.** The resultant graph, \( G_{RA} \), produced by the approximation algorithm is connected if, and only if, the graph \( G_{ADD\_NODE} \) produced by the algorithm ADD_NODE is connected.

**Proof.** For necessary condition, suppose \( u \) and \( v \) are any particular but arbitrarily chosen nodes in \( G_{RA} \). Then by line 2 of the approximation algorithm, the algorithm will consider only those edges that are connected in the graph \( G_{ADD\_NODE} \). Therefore, nodes \( u \) and \( v \) are connected and this is what to be shown.

For sufficient condition, assume to the contrary, that there exists at least one edge in \( G_{ADD\_NODE} \) that is connected i.e. edge \((u, v) \in G_{ADD\_NODE} \) and not connected in \( G_{RA} \) i.e., edge \((u, v) \notin G_{RA} \). Consider the edge \((u, v) \) with minimum range assignment i.e., Euclidean distance \( dist(u, v) \), among all pairs of nodes \( u \) and \( v \) that are connected in \( G_{RA} \) but not connected in \( G_{ADD\_NODE} \). Recall, by definition \( edge(u, v) \in E_{RA} \) iff \( R(u) \geq dist(u, v) \) so, minimum range assignment means the edge connecting node \( u \) and node \( v \) in the \( G_{ADD\_NODE} \). The node \( u \) and \( v \) must be connected directly by edge \((u, v) \) in \( G_{ADD\_NODE} \) (see line 2 of ADD_NODE algorithm). Otherwise, a different edge \((x, y) \) must exist on the path connecting nodes \( u \) and \( v \) would have a Euclidean distance (range assignment) \( dist(x, y) \) less than \( dist(u, v) \) and nodes \( u \) and \( v \) would not be the edge with minimum range assignment i.e. minimum Euclidean distance, \( dist(u, v) \). Since the edge \((u, v) \) is in \( G_{ADD\_NODE} \) by line 2 of the algorithm ADD_NODE, the cost of the shortest path connecting nodes \( u \) and \( v \) is their Euclidean distance i.e. \( RA = dist(u, v) \). Since \( dist(u, x) \) is less than \( dist(u, v) \). Also, since \( dist(u, v) \) is the least such value for any non-connected nodes in \( G_{RA} \) and \( u \) and \( x \) must also be connected in \( G_{RA} \) (by line 2 in approximation algorithm). For the same reason, node \( v \) and node \( x \) must be connected in \( G_{RA} \). This contradicts the supposition that nodes \( u \) and \( v \) are not connected in the graph \( G_{RA} \) produced by algorithm approximation. Hence, the supposition is false and the
sufficient condition holds. And this completes the proof.

6.5 Open Question

The goal of our proposed scheme is to discard long-distance communication links and instead create small links (range assignments). For this purpose, the algorithm chooses a location of additional nodes or relay node using the notion of Steiner tree construction. Clearly, we cannot throw away nodes that are too far from neighbor nodes otherwise, the resultant graph will be partitioned. In general, there is a trade-off between network connectivity and sparseness. Recall, sparseness means that the number of edges in the graph should be in the order of the number of nodes. Although the graph constructed by the approximation algorithm has a nice feature of being at most twice the cost of an optimal cost while maintaining the connectivity, the question of whether the constructed graph is actually sparse remains open. Despite the observation that there exists a node placement and transmitting range setting such that some long-distance links can be removed from the resultant communication graph without increasing the time of the algorithm. We strongly suspect that answer to the question is negative (at least in the context of our problem setting). Our argument is as follows. There always exists a node configuration and transmitting range setting such that the number of communication links in the graph constructed by ADD_NODE algorithm equals to the number of links in the resultant graph.

6.6 Summary

The chapter discussed the algorithmic issues concerning the problem of introducing additional nodes (relay nodes) in the existing wireless network from practical viewpoint. That is, setup a communication links in the emergencies by introducing additional or relay nodes. The chapter also presented a 2-approximation algorithm to assign the transmitting ranges in such a way that the resultant network is connected.
CHAPTER 7

GEOMETRIC ROUTING PROBLEM

7.1 Introduction

In this chapter, we consider the problem of routing in an ad hoc networking for which hosts know nothing about the network except the locations of the hosts to which they can communicate directly. In particular, we consider the case in which all hosts have the same broadcast range. Our algorithm is a geometric routing algorithm for which we give a constructive lower bound. In the literature algorithms of this nature are known as geographic, location-based, and position based algorithms. Formally, given a Euclidean graph \( G = (N, E) \) with \(|N| = n\), the goal of our geometric algorithm is to convey a message from a source \( s \in N \) to a destination \( t \in N \) by sending information packets over the edges of graph \( G \) while obeying the following conditions. Firstly, at the start, all nodes \( v \in N \) in \( G \) know their geometric positions (coordinates) and geometric positions (coordinates) of all of their neighbor-nodes within transmission range \( R \). Note that this assumption becomes more and more realistic with the introduction of inexpensive and miniaturized positioning systems. Secondly, the source node \( s \in N \) knows the position i.e. coordinates, of the destination node \( t \in N \) in the given graph \( G \). For this assumption, a (peer-to-peer) overlay network could be employed \([118,119]\). Lastly, the nodes \( v \in N \) are not allowed to store anything except for temporarily storing packets before transmitting them to the neighbor node(s). Furthermore, a packet can store the information not larger than \( O(\log n) \) bits, i.e. information about \( O(1) \) nodes is allowed. Overviews of geometric routing algorithms are given in \([118,120,121]\). The published results in the references \([1,10–12]\) are the basis of the chapter.

7.1.1 Our Contribution

The chapter presented a fully distributed algorithm to compute a planar subgraph of the underlying wireless connectivity graph. This work considered the idealized unit disk graph model in which nodes are assumed to be connected if, and only if, nodes are within their transmission range. In other words, a graph formed from a collection of equal-radius circles, in which two circles are
connected by an edge if one circle contains the center of the other circle. The main contribution of this work is a fully distributed algorithm to extract the connected, planar graph for routing in the wireless networks. The communication cost of the proposed algorithm is $O(d \log d)$ bits, where $d$ is the degree of a node. In addition, the chapter also presented a geometric routing algorithm and established its lower bound. The algorithm is fully distributed and nodes know only the position of other nodes and can communicate with neighboring nodes in their transmission range.

7.1.2 Chapter Guide

Section 7.2 formulates the problem of extracting the connected planar subgraph from the given graph for the geometric routing.

Section 7.3 discusses the background information for the construction of subgraph for routing.

Section 7.5 presents the overall strategy of the proposed algorithm. For the ease of the presentation, we divided the strategy into two distinct phases.

Section 7.6 presents the proposed technique to construct the planar graph for the geometric routing.

The section presents the formal algorithm, which is fully distributed and employs to extract the connected, planar the sub-graph from the given communication graph. In addition, we discuss some properties of the proposed scheme.

Section 7.7 presents an algorithm for routing in planar graph obtained in the preceding section.

Section 7.8 presents lower bound.

Section 7.9 contains the summary of the chapter.

7.2 Problem Formulation

The chapter investigates the problem of extracting the connected, planar subgraph from the given graph for geometric routing algorithms in ad hoc wireless networks. The preliminary versions of this work appeared in References [11, 12]. In a typical ad hoc network, most of the nodes are mobile but here we assume that routing takes place much faster than node movement. In other words, mobility is not considered here. In addition, we assume that the routing layer can access the location information about network nodes. We shall formulate this problem in the geometric graph as follows. Let $N$ be a set of nodes deployed in a certain region $R$, with $|N|=n$. The problem is to build a planar graph $G=(N,E)$ on $N$ such that each node is connected to its closest neighbors. Formally, the edge $(u, v)\in E$ if, and only if, $\delta(u, v)\leq 1$, where $\delta(u, v)$ is the distance between node $u$ and its closest neighbor $v$. 

In addition, we present a distributed algorithm for routing on the unit disk graph. While the fundamental approach is based upon the famous Face Routing algorithm [46] (in the original paper the algorithm is called Compass Routing II), the main difference is that during traversal of $G$ when the algorithm crossed by the line segment joining the source and the destination nodes, it moves to the adjacent face as suggested in [38].

7.3 Background

Since our algorithm is based on the construction of localized Delaunay triangulation presented in [69,70] and for completeness we outline the method here. After gathering the identity and location information of the nodes, the algorithm finds the Gabriel edges as follows: each edge $uv$ is a Gabriel edge if both angles opposite to $uv$ are less than $\pi/2$. Each node finds all triangles such that all three edges of the triangles have at most one unit length. If the angle at the node is at least $\pi/3$, node $u$ is proposed to form the Delaunay triangulation. If two nodes either accept the proposal or send the proposal, the edge $uv$ is in the Delaunay triangulation. Li et al. [69,70] have shown that in each triangle one of its interior angle is at least $\pi/3$ and form a 1-localized Delaunay triangle. In addition, they showed that if angle at the node is at least $\pi/3$, then the node broadcast at most 6 proposals to construct Delaunay triangulation. The total cost of this construction is $O(n \log n)$ bits. For details and associated proofs, readers may consult [69,70].

7.4 Relevant Literature

To the best of our knowledge, Takagi [122] is the first who proposed geometric routing. Specifically, Takagi solved the problem of the maximum expected progress per hop in the network with the condition that the optimal transmission probability of nodes is in some specified range, say $N$. The paper [122] also talked about the various consequences, especially performance, on a couple of computational models. After this first proposition of geometric routing, we have seen quite a number of works such as [123] and [124] that proposed algorithms based on various notions of progress of information packet or message from source to destination. The main problem with those algorithms was that most of them do not necessarily guarantee message delivery. Another problem, which needs attention, is congestion. As an example, in flooding routing algorithms [125], multiple redundant copies of the messages are sent in the hope that one of them will eventually reach its destination. Clearly, sending multiple copies of messages creates the problem of network congestion. The early
algorithms for routing in ad hoc networks used the greedy approach \[122, 124, 126\]. The basic idea of those algorithms was as follows. A current node forwards an information packet to its neighbor node that is closest to the destination node. However, most of those algorithms failed to deliver a packet to the destination and fall into a local minimum because of the lack of “better” neighbors.

To the best of our knowledge, the reference \[46\] presented the first “correct” Face Routing algorithm that guaranteed packet delivery. The main idea of the Face Routing algorithm is that an information packet walks along faces of planar graphs and proceeds along the line connecting the source and destination nodes. Since we borrowed the basic idea from the famous Face Routing algorithm introduced in \[46\] (in the original paper the algorithm is called Compass Routing II) and for completeness we describe the algorithm here. Start at source \(s\) and let \(F\) be the face that is intersected by line segment joining source \(s\) and destination \(t\), \(\overline{st}\). Explore the boundary of face \(F\) by traversing the edges of \(F\) and remember the intersection point \(p\) on line \(\overline{st}\) with the edge of \(F\) which is nearest to destination \(t\). After traversing all edges, go back to \(p\). If we reach the destination while traversing the boundary of \(F\), we are done. Otherwise, \(p\) divides the line \(\overline{st}\) into two line segments where line \(\overline{pt}\) is not yet “traversed” part of line \(\overline{st}\). Update face \(F\) to be the face which is incident to \(p\) and which is intersected by the line segment \(\overline{pt}\) in the region of \(p\) and start all over again. There is another method proposed in the same paper \[46\] (in the original paper the algorithm is called compass routing) and shown to work for some important classes of networks. Specifically, Kranakis et al. \[46\] have shown that the compass routing works correctly for Delaunay triangulation. The key idea behind the compass routing algorithm is as follows: If an information packet wants to reach the destination node, \(t\), from the source node, \(s\). Then the algorithm will send the packet to the neighbor \(u\) of \(s\) such that the slope of the line segment joining \(u\) to \(s\) is the closest to the slope of the line segment joining \(s\) to \(t\). The problem with compass routing is that it may occasionally fall into infinite loops and as a result the information packet fails to reach the destination. But, this problem never occurs in case of networks whose underlying graph is a Delaunay triangulation.

To the best of our knowledge, Bose et al. \[38\] were the first who presented average-case efficient routing algorithm with guaranteed delivery in ad hoc wireless networks. In fact, these are the first distributed algorithms that do not require duplication of information packets or any kind of duplication of memory at the nodes. Furthermore, the distributed algorithms only require a constant amount of information along with the actual message from source node to destination node. Moreover, the paper presented a simulation to measure the quality of the paths described by the
algorithms. The authors choose to compare their algorithms with distributed algorithm described by Lin et al. [127]. In my opinion, this choice is quite unusual since the algorithm in [127] fails in the case when the information packet consecutively crosses the same edge of the graph twice. Our work is inspired by and based upon the idea articulated in this paper [38]. That is, the idea of finding a planar graph and then applying routing algorithm on this extracted graph.

To the best of our knowledge, the first worst-case analysis of geometric mobile ad hoc routing was presented by Kuhn et al. [128]. In particular, Kuhn et al. presented the first adaptive face routing algorithm with the cost bounded by a function of the optimal route. For example, if a best route has a cost \( c \) (say), then the adaptive face routing finds a route and terminates with cost \( O(c^2) \) in the worst case. In addition, they also gave a tight lower bound by showing that any geometric routing algorithm has worst-case cost \( \Omega(c^2) \). The lower bound analysis presented in the present work is based on the idea presented in Kuhn et al. [128].

Other geometric routing algorithms [38,129], that guaranteed to find the destination on triangulations or convex subdivisions [130], have been suggested. The algorithm in [38] used the Delaunay triangulations as an underlying graph to find the destination node. Consequently, local approximation of the Delaunay Graph has been suggested [45], providing however, no better bound on the performance of routing algorithms in [38,129–131].

The first algorithm that is competitive with respect to the shortest path between the source and the destination was proposed in [128]. This algorithm basically enhances Face Routing by the concept of a bounding ellipse restricting the searchable area. The lower bound example was shown in [128] to be asymptotically optimal.

A more detailed discussion of geometric routing can be found in [132].

7.5 Overall Strategy of the Algorithm

For ease of the presentation, the overall strategy of the algorithm can be divided into two distinct phases. The phase I extracts the connected and planar graph from the given graph while phase II does the actual routing on the graph produced by phase I.

In the phase I, we proposed the distributed algorithm, which is based on [69], on unit disk graph model. The basic idea of the algorithm is as follows. Each node in the given graph, \( G \), broadcasts its identity and position (coordinates) and gathers the identities and positions of their neighbor nodes. Using this information, each node \( u \) computes the local Delaunay triangulation, \( LDG \), such that edges
of the triangles are not larger than one unit. This part of the algorithm is based on the distributed algorithm proposed in [69]. Now each node sends the message to its neighboring node to remove the edges which are not Gabriel edges. When node $u$ receives a message $\text{REMOVE}(\text{edge}(u, v))$, it accepts if there is no point (some node) lies in the disk of diameter $\|uv\|$ otherwise, rejects it by sending the message $\text{REJECT}(\text{edge}(u, v))$. For example, if node $u$ sees that there is a point (some node) in the intersection of its closed neighborhood, $\mathcal{N}(u) \cup \{u\}$, and the disk of diameter $\|uv\|$, then node $u$ sends the message to node $v$ (along with other neighbors) to remove the edge $(u, v)$, $\text{REMOVE}(\text{edge}(u, v))$. Similarly, node $v$ sends the message to remove edge $(v, u)$ to node $u$ (along with other neighbors), if it sees that there exists some node in $(\mathcal{N}(v) \cup \{v\}) \cap \text{disk}(v, u)$. If $u$ and $v$ both sent and received the message $\text{REMOVE}$, then the edge $(u, v)$ will be removed. In other words, if node $u$ has sent the remove message to node $v$ and also received the remove message from node $v$, then the edge $(u, v)$ is removed from the local Delaunay graph, $\text{LDG}$. Since $\text{DT}$ is planar and $\text{GG}$ is connected, therefore, the graph produced by the algorithm is planar and connected.

In the phase II, we will run the routing algorithm based on the Face algorithm [46] (see Section 7.4 for detailed description) on the graph computed by the algorithm in the phase I.

![Figure 7.1: $\text{LDel}^1$ is not a planar graph.](image)

### 7.6 Planar Graph Construction

This section presents the fully distributed algorithm to extract the connected, planar graph. This algorithm also appeared in [12] and the informal version appeared in Reference [11]. The idea of the construction of the planar subgraph is based upon [69]. Li et al. [69] proposed a localized algorithm that constructs a sequence of graphs, called localized Delaunay $\text{LDel}^{(k)}\{N\}$. Our proposed algorithm starts with Li et al. localized algorithm to construct local Delaunay triangulation and
systematically remove and add edges such that resultant graph is planar and connected. Note that neither our proposed algorithm nor Li et al. algorithm guaranteed planar graph. That is, in both of the instances, two local triangles may intersect or the local triangle and Gabriel edge may intersect. As an example, consider the Figure 7.1 in which \( LDel^1 \) is not a planar graph i.e. we have a intersection between local triangle \( uvw \) and Gabriel edge \( xy \). Li et al. [133] solved this problem as follows (depicted in Figure 7.2). A node \( u \) looks for a node in every local triangle. If node \( u \) found such a node, node \( u \) discards the triangle. At the end, node \( u \) keeps the edge if it is a Gabriel edge or, if a triangle belongs to \( LDel^1 \). On the other hand, we propose to solve this problem as follows. A node \( u \) searches for a node in every local triangle. If a node \( u \) found such a node, node \( u \) discard all edges in the neighborhood (depicted in Figure 7.3) and construct localized Delaunay triangulation in the neighborhood. At the end of our proposed solution, the node \( u \) has Gabriel edge and all triangles i.e., edges belong to non-intersecting triangles (depicted in Figure 7.4).

We formalize the idea presented in Section 7.5 in the distributed algorithm as follows.
1 Each node $u$ broadcast its identity and listens to the messages from other nodes
2 Each node $u$ computes the $DT$ with its neighbors (with $N[u] = N(u) \cup \{u\}$)
3 For each edge $uv$, such that disk with diameter $\|uv\|$ contains no other points of $N$. Let
4 $\triangle uvw$ and $\triangle uxz$ be two triangles with common edge $uv$
5 for each edge $u \in N(u) \cup \{u\}$ do
6     Find all triangles $\in Del(N(u) \cup \{u\})$ such that $\|uv\|$, $\|vw\|$, $\|uw\| \leq 1$
7     if $disk(u, v)\cap((N(u)\cup u)\setminus\{u, v\}) \neq \{}$, then
8         Node $u$ broadcast a message PROPOSAL(edge($u, v$)) to remove edge($u, v$) from
9             Delaunay triangle $\triangle uvw$
10        Node $u$ listens to the message from other nodes
11 end
12 if node $v$ receives a message PROPOSAL(edge($u, v$)) then
13     Node $v$ checks the existence of some node in its closed neighborhood i.e.,
14     if $disk(u, v)\cap((N(v)\cup v)\setminus\{u, v\}) \neq \{}$ then
15         Node $u$ broadcast the message ACCEPT(edge($u, v$))
16             /* $u$ accepts the proposal to removing edge($u, v$) */
17     else
18         Node $u$ broadcast the message REJECT(edge($u, v$))
19             /* $u$ rejects the proposal to removing edge($u, v$) */
20 end
21 if edge($u, v) \in Del(N_1(u) \cup \{u\})$ AND $[u$ and $v$ sent REJECTS(edge($u, v$))] then
22     Node $u$ deletes the edges ($u, v$) from its set of incident edges.
23 end

Algorithm 7.1: Connected Graph

Figure 7.3: Our proposed solution: Discard all edges in the neighborhood.

The basic technique here is to find the intersection of $UDG$ and local Delaunay triangulation
(see lines 6 and 13 of Algorithm 7.1). This technique was proposed in [38] in which Bose et al. used
a Gabriel graph with unit disk graph and showed that the minimum spanning tree is a subgraph of
the unit graph. Since, $MST(N)$ is a subset of $DT(N)$, we can apply the same technique [38] to
show that the resultant subgraph is connected if the intersection of unit disk graph and the graph produced by the Algorithm 7.1 is connected. Hence the following lemma.

**Lemma 7.1.** If the unit disk graph of the given set, $UDG(N)$ is connected, then $LDel(N) \cap UDG(N)$ is connected.

**Proof.** Let’s take the negation of the lemma and suppose it to be true. That is, there exists an edge $(u, v) \in MST(N)$ such that $\|uv\| > 1$. Removing the edge $(u, v)$ from $MST(N)$ splits the graph into two connected components $C_1$ and $C_2$ such that $u \in C_1$ and $v \in C_2$. By definition, $UDG(N)$ is connected, it means that there must exists some edge $(w, x)$ with $\|wx\| \leq 1$ such that $w \in C_1(N)$ and $x \in C_2(N)$. If the edge $(u, v)$ is replaced by the edge $(u, v)$, then the resultant graph on $N$ would be connected with weight less than $MST(N)$, a contradiction. Thus, the graph produced by the algorithm is connected.

The following lemma, which is based on [69], establishes the relation between the subgraph produced by the algorithm and the *Gabriel graph* i.e. the edges computed by the algorithm are in fact *Gabriel edges*.

**Lemma 7.2.** If the edge $(u', v')$ computed by the algorithm intersects a local Delaunay triangle $(u, v, w)$, then either $u'$ or $v'$ is inside the disk $(u, v, w)$.

**Proof.** Let triangle $(u, v, w)$ be a triangle with circumcenter $c$, then by definition, at least one of the $u, v,$ or $w$ must be on the different side of the line $u'v'$ with center (say $u$). If both $u'$ an $v'$ are outside of circum circle defined by $(u, v, w)$, then $\angle v'u'u > \pi/2$. Therefore, $u$ is inside the disk $(u, v)$, which is contradicts that $u'y'$ is a Gabriel edge.
Now we show that the graph produced by the algorithm is planar.

**Lemma 7.3.** *The Algorithm Connected Graph produces a planar graph* $G$.

**Proof.** Assume that there exists an edge $(x, y)$ such that $(x, y)$ intersects triangle $(u, v, w)$. Now, edge $xy$ is either a *Gabriel edge* or an edge of a *Delaunay triangle*. If $xy$ is a *Gabriel edge*, then either $x$ or $y$ is inside the disk $(u, v, w)$. On the other hand, if $xy$ is an edge of a *Delaunay triangle*, then either $x$ or $y$ is inside the disk $(u, v, w)$. In any event, there exists one endpoint (either $x$ or $y$) with is outside of disk$(u, v, w)$. This means that $x$ (say) is inside the closed neighborhood, $\mathcal{N}(u) \cup \{u\}$, and $y$ (say) outside the closed neighborhood, $\mathcal{N}(u) \cup \{u\}$. This implies that there is a path from inside the closed neighborhood, $N(u) \cup \{u\}$, to the outside of the neighborhood (since algorithm produces connected graph), a contradiction. Thus, there is no edge in the resultant graph that intersects triangle $(u, v, w)$. Since we have no intersecting edges in the resultant graph, the graph is planar by definition. This completes the proof. □

Although, the above lemma showed independently that the resultant subgraph produced by the algorithm is planar, we can deduce the planarity of the resultant subgraph from Lemma 7.1 as follows. Since $UDG$ is planar and the *local Delaunay triangulation* is planar, therefore, the intersection of $UDG$ and *Delaunay triangulation* must be planar.

The subsequent line of arguments is based on the Li et al. [69]. For brief introduction see Section 7.5. Since a graph produced by the algorithm is planar by Lemma 7.3, and a proposal made only if $\angle wuv \geq \pi/3$, node $u$ broadcasts at most 6 proposals. And each proposal is replied to by at most two nodes. Therefore, the total communication cost is $O(d \log d)$ bits, where $d$ is the degree of a node.

### 7.7 Geometric Routing

In this section we describe an algorithm for routing in a planar graph obtained in the previous section. The informal versions of this algorithm appeared in [11,12]. The algorithm is based on [46] and the basic idea is as follows. Let $f$ be the face of $G$ with a starting point $p$ on its boundary that intersects line segment $(p, t)$, where $t$ is the destination. Using right-hand rule, traverse the face $f$ in the counterclockwise direction. If the edge $(u, v)$ of the face $f$ intersects with $(p, t)$ at $p'$ and $\text{dist}(p', t) < \text{dist}(p, t)$ then this intersection $p'$ becomes the new starting point $p$. In the similar fashion traverse faces until $p$ becomes a destination $t$. We used the same idea but instead
interchanging the points, the traversal moves to the adjacent face crossed by line segment \((p, t)\) as suggested in [38]. Note that even though we have not mentioned it explicitly in the algorithm, the traversal is accomplished by right-hand rule. See Section 1.1 of the Chapter 1 for the relation between traversal and right-hand rule.

\[
\begin{array}{l}
  p = s \\
  \text{for all edge } (u, v) \in f \text{ do} \\
  \quad \text{if } u, v \text{ intersects } (p, t) \text{ at a point } p' \text{ then} \\
  \quad \quad \text{if face } f \text{ contains } p \text{ and } p = t \text{ then} \\
  \quad \quad \quad \text{we are done} \\
  \quad \quad \text{else} \\
  \quad \quad \quad \text{move to the adjacent face} \\
  \quad \text{end} \\
  \text{end}
\end{array}
\]

Algorithm 7.2: Greedy Traversal

For the sake of the argument (adversary argument), we say that the above algorithm might drop the packet if a point, \(p'\), for a face change is not found after the complete face traversal. We claim that since underlying graph is planar, this situation will never occur provided the destination is connected. Reader may refer to the Section 1.1 of the Chapter 1 for details. In particular, since underlying subgraph is planar by Lemma 7.3 and the destination is connected (implied by Lemma 7.1), the face-change rule must always find a face closer to destination than the current face along line segment joining source-destination. See brief introduction of face routing [46] in the Section 7.4.

The subsequent two lemmas establishes the correctness by showing the forward progress of the packet and that the algorithm will terminate after executing a finite number of steps.

**Lemma 7.4.** Algorithm Greedy-Traversal reaches destination \(t\) in a finite number of steps.

Since the underlying graph is connected and planar by Lemma 7.3 and since the distance to the destination is decreasing during each pass of the algorithm, therefore, algorithm Greedy-Traversal reaches destination \(t\) in a finite number of steps.

Following lemma [46] gives the bound on the number of steps.

**Lemma 7.5.** Algorithm Greedy-Traversal reaches the destination \(t\) after at most \(4|E|\) steps, where \(|E|\) is the number of edges in \(G\).

The key to the proof of Lemma 7.5 is that the algorithm must go across the entire face \(f\) to
determine the point of intersection and then must return to the intersection point. For proof, reader may consult [46]. However, to emphasize the importance of Lemma 7.5 in the context of the geometric algorithms, we like to add interesting point regarding the bound appearing in the Lemma 7.5. That is, Bose et al. [38] suggested the way to reduce the bound $4|E|$. They proposed that the bound $4|E|$ of the face routing [46] can be reduced to $3|E|$ by forcing the algorithm to follow the shortest path when returning to intersection points. Although we believe that this “cunning” observation of Bose et al. [38] would increase the performance of any face routing algorithm [38, 46, 127, 132] in practice especially, in the case of present work, we are not sure what consequence this might have on the algorithmic complexity. Particularly, how this “shortest path” calculation influence the overall complexity of the algorithm. We hope to extend the present work by including this fine point of Bose et al. [38].

7.8 Lower Bound Analysis

This section presents a lower bound for the routing algorithm running on the connected planar graph constructed in Section 7.6. We formalized the reasoning presented in [128] to establish the lower bound of the algorithm.

**Lemma 7.6.** The algorithm presented in Section 7.6 has the cost $\Omega(n^2)$ where $n$ is the number of nodes in $G$.

**Proof.** Suppose $G$ be the Euclidean graph $G$ and $2k$, where $k$ is positive integer, nodes are evenly distributed on a circle. Since we are working on $UDG$, it is reasonable to suppose (without loss of generality) that the Euclidean distance between neighboring nodes is exactly 1. There are routes (paths) from the nodes on the boundary of circle towards the node at the center of the circle such that the distance between adjacent nodes is exactly 1. Now, let $s$ be the particular but arbitrarily chosen node on the circle boundary such that there exists a route (path) from node $s$ to the node $t$ at the center. The algorithm finds the route (path) from the source node $s$ to the destination node at the center. As shown in [128], an optimal route between source and the destination follows the shortest path on the circle until it hits the node $s$ and then directly follows $s$’s path to $t$. The cost of such link is not larger than the number of nodes in a circle of given radius i.e. $c \leq k+r+1$. In other words, the cost of the optimal path from source $s$ at the circle boundary to destination $t$ at the center is $O(n)$. In this scenario, the goal of the routing algorithm is find the correct path that leads to the destination node $t$. Since according to our assumption, there is no routing information
stored at the node, the only way to find such a path is search exhaustively for the required path. Therefore, the algorithm has to explore \( n^2 \) nodes in the worst case. Because the underlying graph is a unit disk graph with edge length exactly 1 (according to our assumption) leads to the cost \( \Omega(n^2) \). This completes the proof.

We have shown that in the worst the case our algorithm is quadratic in the cost of an optimal path in the given graph \( G \).

7.9 Summary

We have presented a technique to extract the connected, planar subgraph for geometric routing algorithms. We considered the idealized unit disk graph model in which nodes are assumed to be connected if, and only if, nodes are within their transmission range. The main contribution of the chapter is a fully distributed algorithm to extract the connected, planar graph for routing in the wireless networks. The communication cost of the proposed algorithm is \( O(d \log d) \) bits, where \( d \) is the degree of a node. We have also presented the geometric routing algorithm that is based upon the famous Face Routing algorithm. The algorithm is fully distributed and nodes know only the position of other nodes and can communicate with neighboring nodes in their transmission range. We have shown that in the worst case our algorithm is quadratic in the cost of an optimal path in the given graph.
CHAPTER 8

CONCLUSION AND FUTURE DIRECTION

In the Chapter 4, we presented a parallel algorithm that utilizes the local search technique for computing a Steiner tree in the two dimensional plane. We implemented the local search technique in parallel for Steiner tree problem that allows us to solve larger problems and improve the quality of the final solution. The main contribution of this work is the $O(n^2 \log^2 n+\log n \log(n))$ parallel local search algorithm for computing a Steiner tree on the Euclidean plane. The main advantage of the algorithm is that it does not need synchronization. As a result, it has no communication overhead.

We have presented a simulation of the parallel Steiner tree problem on the server-client model of computation. Particularly, we have analyzed the time delays at server end and examined some consequences of time delays on the execution time of the parallel Steiner tree algorithm. In addition, we have also computed the expression for the optimal number of client processors required by the parallel algorithm for optimal efficiency.

The Chapter 5 addresses the problem of constructing the planar Voronoi diagram in parallel on the hypercube model of computation. While the more novel results have been in the area of parallel algorithms, the real focus of the chapter is to combine theoretical and practical results to develop a practical parallel algorithm by exploiting the characteristic properties of the computational model. In the chapter, we have shown that the problem of constructing planar Voronoi diagram can be solved using hypercube model of computation with $n$ processors in $O(lg^2(n))$ time. The fact that the bitonic sort algorithm bounds the time complexity of the algorithm also showed that sorting helps in the constructing of the Voronoi diagram.

The fundamental problem in the constructions of the sequential Voronoi diagram is the construction of polygonal line quickly. The chapter shows that this fundamental problem remains unchanged even in the parallel setting. One of the features of the algorithm presented is to construct a polygonal dividing chain from intersection points. Hence, we have shown that it is certainly possible to construct the polygonal chain without explicitly computing all parts of it.
The Chapter 6 discussed the algorithmic issues concerning the problem of introducing additional nodes (relay nodes) in the existing wireless network from practical viewpoint. That is, setup a communication links in emergencies by introducing additional or relay nodes. The chapter also presented a 2-approximation algorithm to assign the transmitting ranges in such a way that the resultant network is connected.

In the Chapter 7, we have presented a technique to extract the connected, planar subgraph for geometric routing algorithms. We considered the idealized unit disk graph model in which nodes are assumed to be connected if, and only if, nodes are within their transmission range. The main contribution of the chapter is a fully distributed algorithm to extract the connected, planar graph for routing in the wireless networks. The communication cost of the proposed algorithm is $O(d \log d)$ bits, where $d$ is the degree of a node. We have also presented the geometric routing algorithm that is based upon the famous Face Routing algorithm. The algorithm is fully distributed and nodes know only the position of other nodes and can communicate with neighboring nodes in their transmission range. We have shown that in the worst case our algorithm is quadratic in the cost of an optimal path in the given graph.

Mobility is one of the most important features of an ad hoc wireless networking, but we have limited the scope of the present work by assuming that the routing was taking place much faster than nodes movement. The inclusion of node(s) movement is the next logical topic for the future research. In addition, we hope to extend our work beyond the flat topology. A promising direction is to work in 3D-volumes rather than 2D-faces.
BIBLIOGRAPHY


