The Euclidean Arborescence Problem

A Thesis Submitted to the College of Graduate Studies and Research in Partial Fulfillment of the Requirements For the Degree of Master of Science in the Department of Computer Science University of Saskatchewan Saskatoon, Saskatchewan

by

Bradley W. Mazurek

Fall 1996

© Copyright Bradley W. Mazurek, 1996. All rights reserved.
Permission To Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 0W0
Abstract

The Euclidean arborescence problem involves the creation of rooted trees embedded in the plane using the L₂ distance metric. These trees are interesting in that they have a low cost yet offer responsive service from the root to any other vertex. As such, arborescences have their cost compared to that of the minimum spanning tree (MST), and their radius compared to that of the shortest path tree (SPT), which are minimal with respect to cost and radius, respectively. This research examines geometric techniques for constructing such arborescences.

The central component to this research is the development of a generalized arborescence algorithm framework. Independent framework modules are used to define a unique arborescence algorithm. Arborescence properties are defined, including metrics to measure the quality of the arborescence relative to the MST cost and the SPT radius.

This framework is used to describe four arborescence algorithms: the circle spanning, circle Steiner, unrestricted tangent and restricted tangent arborescence algorithms. Each algorithm is analyzed to determine its computational complexity and space requirements, as well as its theoretical performance with respect to the described quality metrics.

However, because theoretical bounds for metrics are not always available or reflective of practice, empirical research was done to determine how each of the algorithms perform in practice. The results of this experimentation look quite favorably on three of the four arborescence algorithms considered. The experimental values for those three have very stable and predictable quality on large point sets, which is well under a two approximation on most metrics.
Acknowledgements

There are countless people who deserve my utmost gratitude for their role in helping me complete my Master's degree. However, some I must thank individually:

My supervisor, Mark Keil, for his help and guidance over the past two years. His patience and encouragement were extraordinary, especially during the latter stages of thesis writing.

The members of my committee: Professors Grant Cheston, Anthony Kusalik and Raj Srinivasan. Their comments and constructive criticisms have dramatically improved the quality of my thesis.

Kevin Froese and Fabian Searwar, for the friendship, encouragement, and suggestions over the past many years. Greg Oster, for his unselfish willingness to help others, whatever the circumstance. Martin Arlitt, Leanne Breker, Tim Harrison, Orland Roeber, Lee Kennedy and Greg Oster for their friendship and support.

Sean Weisensel and Scott Jasken, not only for being great roommates, but for their friendship and encouragement as well.

The University of Saskatchewan, for the Graduate Scholarship which funded my post-graduate education.

My parents, Ray and Jean, and my siblings, Kevin and Darla, for their love and inspiration, and for teaching me to believe in myself. And to Michelle Eldridge, for her unconditional love and support over the past many years.
FOR MY PARENTS
# Table Of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permission To Use</td>
<td>i</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>iii</td>
</tr>
<tr>
<td>Table Of Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>ix</td>
</tr>
<tr>
<td>List of Figures</td>
<td>x</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Background</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Definitions</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Motivation</td>
<td>11</td>
</tr>
<tr>
<td>2.2.1 Service Delivery</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2 Multicast Routing</td>
<td>13</td>
</tr>
<tr>
<td>2.2.3 VLSI Design</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Previous Work</td>
<td>15</td>
</tr>
<tr>
<td>2.3.1 Rectilinear Steiner Arborescences</td>
<td>15</td>
</tr>
<tr>
<td>2.3.2 Prim-Dijkstra Tradeoffs</td>
<td>16</td>
</tr>
<tr>
<td>2.3.3 LAST Trees</td>
<td>18</td>
</tr>
<tr>
<td>2.4 Summary</td>
<td>18</td>
</tr>
</tbody>
</table>
### 3 Arborescence Algorithm Framework

- **3.1 Regions of Influence** .................................. 20
- **3.2 The Sweepcircle Technique** .......................... 22
- **3.3 Events** ................................................. 24
- **3.4 Sweepcircle Adjacency Cycles** ....................... 25
- **3.5 The Algorithm Framework** ............................ 27
- **3.6 Independent Framework Modules** .................... 28
  - **3.6.1 Region of Influence Definition** .................. 28
  - **3.6.2 Component Merge Techniques** .................... 29
    - **3.6.2.1 Bounding the Radius** ......................... 32
  - **3.6.3 Not Using the Sweepcircle Adjacency Cycle** .. 34
- **3.7 Data Structures** ..................................... 35
- **3.8 Arborescence Properties** ............................. 37
- **3.9 Summary** .............................................. 39

### 4 Circle Arborescence Algorithm

- **4.1 Region of Influence Definition** ..................... 41
  - **4.1.1 Intersecting Regions of Influence** ............... 42
- **4.2 Circle Spanning Arborescence** ....................... 46
  - **4.2.1 The Algorithm** ................................ 47
  - **4.2.2 Algorithm Analysis** ............................. 48
    - **4.2.2.1 Time Complexity Analysis** ................ 48
    - **4.2.2.2 Space Requirements Analysis** ............... 49
  - **4.2.3 Algorithm Properties** .......................... 50
- **4.3 Steiner Circle Arborescence** ....................... 55
  - **4.3.1 The Algorithm** ................................ 55
  - **4.3.2 Algorithm Analysis** ............................. 57
    - **4.3.2.1 Time Complexity Analysis** ................ 57
    - **4.3.2.2 Space Requirements Analysis** ............... 58
  - **4.3.3 Algorithm Properties** .......................... 59
4.4 Summary .................................................. 62

5 Tangent Arborescence Algorithm 63
5.1 Region of Influence Definition .................................. 63
  5.1.1 Intersecting Regions of Influence .......................... 65
5.2 Spanning Tangent Arborescences ................................ 67
5.3 Unrestricted Tangent Arborescence .............................. 68
  5.3.1 The Algorithm .............................................. 68
  5.3.2 Algorithm Analysis .......................................... 71
    5.3.2.1 Time Complexity Analysis ............................. 71
    5.3.2.2 Space Requirements Analysis .......................... 72
    5.3.2.3 Algorithm Properties ................................. 72
5.4 Point of Intersection Tangent Arborescence .................... 73
5.5 Restricted Tangent Arborescence ................................ 74
  5.5.1 The Algorithm .............................................. 82
  5.5.2 Algorithm Analysis .......................................... 86
    5.5.2.1 Time Complexity Analysis ............................. 86
    5.5.2.2 Space Requirements Analysis .......................... 87
    5.5.2.3 Algorithm Properties ................................. 87
5.6 Summary .................................................. 90

6 Empirical Study 91
6.1 Implementation .............................................. 91
  6.1.1 Metrics .................................................. 94
6.2 Experimental Design ......................................... 96
6.3 Measurements ............................................... 99
  6.3.1 Arborescence Cost ....................................... 99
  6.3.2 Arborescence Radius ..................................... 103
  6.3.3 Arborescence Maximum Vertex Radius ...................... 107
6.4 Arborescence Algorithm Quality ................................ 111
6.5 Summary .................................................. 113
7 Conclusion

7.1 Summary and Contributions .............................................. 114
7.2 Future Work ........................................................................ 118

References .............................................................................. 125
List of Tables

7.1 Theoretical Algorithm Properties ........................................ 116
7.2 Experimentally Observed Algorithm Properties (5000 Vertices, Uniform Distribution) ........................................ 117
List of Figures

1.1 Example Arborescence ........................................... 2
2.1 Voronoi Diagram and Delaunay Triangulation ...................... 7
2.2 Tree Configurations .............................................. 8
2.3 Unbounded SPT and MST Properties ................................. 10
2.4 Example Rectilinear Steiner Arborescence .......................... 16
3.1 Proximity Structures .............................................. 21
3.2 Applications of Regions of Influence ............................... 23
3.3 Sweepline Regions and Their Associated Adjacency Cycles ....... 26
3.4 Spanning Merge .................................................. 30
3.5 Classic Steinerization Merge ...................................... 30
3.6 Finding a Toricelli Point .......................................... 31
3.7 Point of Intersection Steinerization Merge ......................... 32
3.8 Existence of “Pockets” ............................................ 35
3.9 Where radius\(_{\text{ratio}}\) and radius\(_{\text{max\_ratio}}\) Differ ................. 39
4.1 Sigma Parameter Variations ....................................... 40
4.2 Regions of Influence: Circle Arborescence Algorithm ............ 42
4.3 ROI Overall Shape: Circle Arborescence Algorithm ............... 42
4.4 ROI Intersection: Case 1 .......................................... 43
4.5 Ray Shooting in the Voronoi Diagram ................................ 43
4.6 ROI Intersection: Case 2 .......................................... 45
4.7 Circle Spanning Arborescence With Poor cost\(_{\text{ratio}}\) ................. 50
4.8 Effect of Lowering SPT Bound on cost\(_{\text{ratio}}\) ........................ 53
4.9 Circle Steiner Arborescence With Poor \( \text{cost}_{\text{ratio}} \) ........................................ 59
4.10 Maximizing a Local Tree .............................................................. 60

5.1 Two Vertices Become “Visible” to Each Other ........................................ 64
5.2 Defining a ROI for the Tangent Arborescence Algorithm ............................. 64
5.3 Region of Influence: Tangent Arborescence Algorithm .............................. 65
5.4 ROI Overall Shape: Tangent Arborescence Algorithm .................................. 65
5.5 ROI Intersection: Case 1 ............................................................................ 66
5.6 ROI Intersection: Case 2 ............................................................................ 67
5.7 Spanning Tangent Arborescence ................................................................. 68
5.8 Point of Intersection Tangent Arborescence ................................................ 74
5.9 Modified ROI: Restricted Tangent Arborescence Algorithm ......................... 75
5.10 ROI Overall Shape: Restricted Tangent Arborescence Algorithm ................. 75
5.11 Geometry of Modified ROI Intersections .................................................... 76
5.12 Determining \( \overline{BC} \) .............................................................................. 79
5.13 Determining the ROI Intersection Case ...................................................... 81
5.14 Restricted Tangent ROI Shape ..................................................................... 88

6.1 Example Circle Spanning Arborescence ........................................................ 92
6.2 Example Circle Steiner Arborescence .......................................................... 93
6.3 Example Unrestricted Tangent Arborescence ............................................... 94
6.4 Example Restricted Tangent Arborescence ................................................... 95
6.5 Example Minimum Spanning Tree ............................................................... 96
6.6 Example Shortest Path Tree ......................................................................... 97
6.7 Uniform Vertex Distribution ........................................................................ 97
6.8 Normal Vertex Distribution ........................................................................ 98
6.9 Clustered Vertex Distribution ...................................................................... 98
6.10 Circle Spanning \( \text{cost}_{\text{ratio}} \) Over Various Distributions .............................. 100
6.11 Comparing Distributions for Circle Spanning Algorithm ............................ 100
6.12 Algorithm Ranges for \( \text{cost}_{\text{ratio}} \) Metric ...................................................... 102
6.13 Arborescence Quality by Algorithm on a Uniform Distribution ................. 102
Chapter 1

Introduction

In a service-oriented society, consumers demand prompt service and a low price. One of these competing demands is often sacrificed to the benefit of the other. In general, the more responsive a service, the more expensive that service becomes to provide. Similarly, a less responsive service is cheaper to provide. Because costs are passed along to consumers, these tradeoffs between fast/expensive and slow/cheap service are determined by the needs and financial situation of the clients. Perfect examples of such services include the international postal service (cheap, slow) and international couriers (expensive, fast).

Service providers within a region are often centralized: there is one principal distribution point from which services are administered and dispatched. When such providers are devising plans of action, service routes, etc., it is useful to keep these speed versus cost tradeoffs in mind. Mechanisms for providing inexpensive yet prompt service would be very desirable.

Frequently, the model for delivery of such services has been a tree-like structure: service units branch out from the central distribution center, accommodating all sites within a given region. Thus, the service providers wish to construct these service trees such that every client can be reached quickly, but also such that the entire tree is not overly expensive.

Consider the service tree in Figure 1.1. Located in the middle is a relatively large point. This point represents the service center, and every other point represents a client. The lines between points represent a portion of a service route. As stated
above, it is desirable to construct service routes which are both inexpensive yet responsive. The service routes depicted in Figure 1.1 are just such routes, being approximately 12.6% more expensive than the least expensive tree, and 52.9% less responsive than the most responsive tree.

Generally, the system of service routes desired consists of \( n-1 \) edges on \( n \) vertices (\( n - 1 \) clients and one service center) and are called service trees, or simply trees. The single specialized vertex (service center) is called the root, and makes the tree a
rooted tree. The first desirable quality making a rooted tree a service tree is low cost (inexpensive). The cost is determined by summing the lengths of the edges forming the tree. The second desirable quality is responsiveness (low radius). In other words, the distance from any vertex to the root through the tree is small. The radius for a vertex is determined by summing the lengths of the edges on the shortest path in the tree between the vertex and the root. In this thesis, the term arborescence will be used for rooted trees which have both a low cost and a small radius.

This thesis will examine geometric methods and techniques for generating arborescences on arbitrary point sets in the Euclidean plane. The techniques used to construct these arborescences will be examined, and their computational complexity and space requirements will be established. The arborescences produced will be analyzed to determine both theoretical and experimental quality with respect to both cost and radius. Applications of these specialized trees will be examined, especially in the areas such as VLSI design, multicast routing in networks, as well as their use in planning considerations of various service organizations.

In this study of arborescence-generating techniques, a generalized framework for constructing arborescences will be presented. By precisely defining particular modules within the framework, it becomes an arborescence-generating algorithm. Two classes of such algorithms (a total of four algorithms) are described using the framework. These algorithms then undergo both theoretical and experimental quality assessments.

The organization of this thesis is as follows. Chapter 2 will begin with a presentation of definitions and concepts used in computational geometry. It includes a more formal introduction of arborescences, as well as motivation for arborescence study and finally a survey of related previous work. A generalized framework for constructing arborescences and the metrics by which their quality will be judged is presented in Chapter 3. By precisely defining particular procedures within the framework, an arborescence-generating algorithm is obtained. Chapters 4 and 5 present two classes of such algorithms (a total of four algorithms) described using the framework. Theoretical analysis of each is examined in their respective chapters.
Chapter 6 is an empirical study of the four algorithms with respect to the metrics discussed in Chapter 3. Finally, Chapter 7 gives a summary of the work done in the thesis, including discussion of contributions and suggestions for future research directions.
Chapter 2

Background

2.1 Definitions

This chapter begins with a discussion of terms, definitions, structures and algorithms used in computational geometry that are relevant to the task at hand.

The primary object of study is a set of points in the plane under a given distance metric. This metric provides a mechanism to measure distances between pairs of points. The terms point (points) and vertex (vertices) will be used interchangeably.

Euclidean space is a commonly used metric space, in which the location of each vertex is identified by an ordered set of coordinates. In \(d\)-dimensional Euclidean space, a vertex \(i\) would have its location denoted by \((i_1, i_2, \ldots, i_d)\). The distance between any pair of vertices can be determined by the distance metric \(L_p\). For any \(1 \leq p \leq \infty\) the \(L_p\) distance between two vertices \(i\) and \(j\) is expressed as follows: \[19\]

\[
\text{dist}_p(i, j) = \left( \sum_{k=1}^{d} |i_k - j_k|^p \right)^{\frac{1}{p}}
\]

In the two dimensional plane, under the \(L_2\) metric, the distance formula reduces to the familiar \(\text{dist}_2(i, j) = \sqrt{(i_x - j_x)^2 + (i_y - j_y)^2}\), where vertex \(i = (i_x, i_y)\) and \(j = (j_x, j_y)\). The following notation will be used in lieu of \(\text{dist}_2(i, j)\): \(\text{dist}(i, j)\) or \(ij\).

The discussion herein is with respect to 2-dimensional Euclidean space under the \(L_2\) metric unless otherwise noted.
A Euclidean graph is denoted by \( G = (V, E) \), where \( V = \{v_1, v_2, \ldots, v_n\} \) is a set of points or vertices in the plane, and \( E \) is a set of edges between vertices of \( V \). Each is denoted by \( (v_i, v_j) \) where \( 1 \leq i, j \leq n \), and has length \( \text{dist}(v_i, v_j) \). The Euclidean distance between two points, \( v_i \) and \( v_j \) (not necessarily adjacent), will be denoted \( \text{dist}(v_i, v_j) \), and the shortest distance between \( v_i \) and \( v_j \) through any graph \( G \) will be denoted \( \text{dist}_G(v_i, v_j) \).

Additionally, when dealing with a Euclidean graph it is often desirable to have a sense of how large it is. Therefore, the cost of a graph \( G \) will be represented as \( |G| \) and will be defined as follows:

\[
|G| = \sum_{v(v_i, v_j) \in E} \text{dist}(v_i, v_j)
\]

Let \( V \) be a set of \( n \) points in the plane. The complete Euclidean graph \( G_C = (V, E_1) \) is the Euclidean graph in which \( E_1 \) contains an edge between every pair of vertices in \( V \). A Euclidean spanning graph \( G_S = (V, E_2) \) is a Euclidean graph in which there exists at least one path between every pair of vertices in \( V \). Clearly, the complete Euclidean graph is also a Euclidean spanning graph. A spanning tree on \( V \), \( T = (V, E_3) \) is a spanning graph with \( n - 1 \) edges. The only vertices in a spanning tree are those in \( V \).

The Voronoi Diagram is one of the most important structures in computational geometry[17]. Given \( n \) points, a Voronoi Diagram partitions the plane into \( n \) distinct regions, each associated with one of the \( n \) vertices. A point is in the region associated with vertex \( i \) if it is closer to vertex \( i \) than any of the other \( n-1 \) points. Figure 2.1(a) is an example of the Voronoi Diagram for a random set of points.

The dual of the Voronoi Diagram is called the Delaunay Triangulation (DT). The DT of the same point set is displayed in Figure 2.1(b). Many other important proximity structures are derived from the Voronoi Diagram and the DT. Despite the wealth of information provided by the Voronoi Diagram, there is an efficient \( (O(n \log n)) \) algorithm to find it[19].
One of the proximity structures derived from the DT is the Minimum Spanning Tree (MST ⊂ DT). The MST of a set of \( n \) vertices is the least-cost network of edges interconnecting the points using \( n - 1 \) of the \( (n^2 - n)/2 \) edges in the complete Euclidean graph of the vertex set. In other words, it has the smallest cost of all spanning trees for a given point set. Due to its relationship with the DT, finding the MST becomes easier because there are only \( O(n) \) edges in the DT, compared with \( O(n^2) \) edges in the complete Euclidean graph. Therefore, finding the MST requires \( O(n \log n) \) time. Figure 2.2(a) shows the MST for an example point set. The MST on \( \{v_1, v_2, \ldots, v_n\} \) will be referred to as \( MST(v_1, v_2, \ldots, v_n) \).

Although the MST is the least-cost tree connecting \( n \) points using only the edges between them, the Steiner Minimal Tree (SMT) is the least cost spanning tree on \( n \) points. To accomplish this further minimization from the MST, the SMT uses up to \( n - 2 \) additional vertices called Steiner points. These Steiner points are introduced at locations where they will minimize overall tree cost. When necessary, the SMT on \( \{v_1, v_2, \ldots, v_n\} \) will be referred to as \( SMT(v_1, v_2, \ldots, v_n) \). As can be seen in Figure 2.2(b), the introduction of Steiner points can impact the structure of the resulting tree. However, for a general point sets, finding the SMT is an NP-hard
Thus, no efficient algorithms for generating SMTs are known. Instead, approximations and heuristics are often used where the SMT is desired. Obviously, \(|SMT(N)| \leq |MST(N)|\) for a general point set, \(N\), but the MST can be used as a heuristic for the SMT. In 1968, Gilbert and Pollak\([11]\) speculated that

\[
\forall N \quad \frac{|SMT(N)|}{|MST(N)|} \geq \frac{\sqrt{3}}{2} \approx 0.866
\]

This became known as the Steiner Ratio Conjecture and has since been proven correct by Du and Hwang\([10]\).
Another style of spanning tree is the rooted spanning tree. This variation differs in that one vertex in the point set is identified to be the root, \( r \). When so defined, it becomes possible to identify and examine new properties of this spanning tree. These new properties will obviously involve the root in some capacity. For example, the radius of a point \( v_i \) in a rooted tree \( T \) is defined to be the distance between \( v_i \) and \( r \) in the tree. In other words,

\[
\text{rad}_T(v_i) = \text{dist}_T(v_i, r)
\]

Further, let the radius of a rooted tree \( T \) be the maximum of all such radii in \( T \):

\[
\text{rad}(T) = \max_{i=1...n}(\text{rad}_T(v_i))
\]

The Shortest Path Tree (SPT) is a spanning tree on \( n \) points, rooted at a particular vertex \( r \) such that, for any vertex \( v_i \), the distance between \( r \) and \( v_i \) (i.e., the radius of \( v_i \)) in the tree is minimized. Thus, for the complete Euclidean graph, the Shortest Path Euclidean Tree \( T \) simply consists of an edge from each non-root vertex to the root. Or more formally, \( T = (V, E) \) where \( V = \{r, v_1, v_2, \ldots, v_{n-1}\} \) and \( E = \bigcup_{i=1}^{n-1}\{(v_i, r)\} \). Figure 2.2(c) shows the SPT for the same point set as in (a) and (b). The SPT of the set \( \{r, v_1, v_2, \ldots, v_{n-1}\} \) will be referred to as \( SPT(r, v_1, v_2, \ldots, v_{n-1}) \).

With an understanding of the inexpensive nature of the MST and SMT and an understanding of the responsive nature of the SPT will often come a realization of how significantly different they can be with respect to cost and radius. Although the MST and SMT minimize the cost of the resulting tree, the radius of these trees can be arbitrarily large when compared to than the SPT radius. Consider the Minimum Spanning Trees in Figures 2.3(a) and (d). Each has a minimized cost, yet the radius of each tree is very large.

Alternatively, the SPT has a radius which is minimized, but the cost of the tree can be arbitrarily large. In Figures 2.3(c) and (f), two point sets and their associated
SPT can be seen. It becomes clear that the cost of these trees is significantly more than the minimum spanning trees of the same point sets seen in Figures 2.3(a) and (d). However, the radius of a MST can be arbitrarily large. Fortunately, there do exist trees which have a low cost while having a small radius, such as those seen in Figures 2.3(b) and (e). Neither tree is minimum in cost or radius, but both retain near-optimal values of each metric. The term *arborescence* will be used to describe rooted trees embedded in the plane which have both a low cost and a small radius.

The term *arborescence* has been previously used to describe other, similar styles of rooted trees. Trees in the plane under the $L_1$ metric with all edges directed away from the root have been called arborescences\[21\], and are used in VLSI design. In graph theoretic realms, an arborescence is a rooted tree with directed edges leading away from the root. This thesis will examine arborescences in the plane where edges are undirected and distances obey the $L_2$ metric.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{arborescence.png}
\caption{Unbounded SPT and MST Properties}
\end{figure}

Consider the trees in Figures 2.3(b) and (e) and Figure 2.2(d). Clearly, these trees are spanning arborescences, yet their cost is still reasonably close to that of the corresponding MST.
As was noted earlier, the introduction of Steiner points further minimize tree cost when comparing the MST to the SMT. Therefore, Steiner points will be introduced to reduce the size of the spanning arborescence. The introduction of Steiner points into an arborescence may reduce the cost of the resulting tree. As shall be seen later, however, Steiner points often help reduce the radius of the arborescence rather than the cost.

When describing the quality of an arborescence $A$ rooted at a vertex $r$, Khuller et al.[15] define the concept of an $(\alpha, \beta)$-tree. For $\alpha \geq 1$ and $\beta \geq 1$, $A$ is an $(\alpha, \beta)$-tree if it meets the two following requirements:

- Distance requirement: For every vertex $v$, $\text{dist}_A(r, v)$ is at most $\alpha \cdot \text{dist}(r, v)$.
- Cost requirement: The cost of $A$ is at most $\beta \cdot \text{MST}(r, v_1, v_2, \ldots, v_{n-1})$.

For example, if $\alpha$ is fixed at 1, the $(\alpha, \beta)$-tree denotes the shortest-path tree with cost at most $\beta$ times the cost of the MST. Obviously the objectives when setting $\alpha$ and $\beta$ low are in direct conflict with each other. These tradeoffs are examined further in Section 2.3.3.

## 2.2 Motivation

Although the study of arborescences is not well known, the tradeoffs between cost and radius (responsiveness), which are being examined here, are frequently of interest to corporations and other organizations. The following is a closer examination of applications for arborescences.

### 2.2.1 Service Delivery

In today's competitive marketplace, the success of a company is determined not only by the quality of services offered, but also by prompt service. The service providers wish to provide fast, prompt service, yet keep delivery costs low. Regardless of the goods delivered or services rendered, the proper mix of prompt and inexpensive service are often pivotal in determining the success of a company.
The various types of parcel delivery services (including the postal service and couriers) provide perhaps the best example of the cost/responsiveness tradeoff experienced by organizations. Each service dispatches units to collect and bring parcels to a centralized site where they distribute the parcels to their destinations. The most inexpensive delivery service, bulk mail, provides slow and unresponsive service. First class mail is somewhat more expensive, but delivery time is much better. Courier services charge a premium fee, but deliver packages very quickly. Using arborescences to describe this model is quite simple. The central site would be the arborescence root, with collection and delivery routes representing the edges. The homes and businesses utilizing the delivery services are represented by the non-root vertices in the point set.

Arborescence-based delivery trees can be introduced between warehouses and retail outlets within a corporation. When delivering delay sensitive (spoilable) goods, such as fruit, vegetables or meat, from a depot to retail outlets, fast yet inexpensive delivery is very desirable. When modeling this system with arborescences, the central warehouse or distribution center is represented by the root. Edges represent delivery routes and, if they exist, Steiner points represent secondary storage facilities. Finally, non-root vertices would represent retail outlets selling goods to consumers.

When designing routes for school buses, tradeoffs between the cost and the length of time for a trip become obvious. To model this situation, the root vertex is analogous to the school, and non-root vertices represent the homes of the school children. Edges represent the bus routes used to move children to and from school. In reality, school boards cannot afford to have many near empty buses or, if taken to an extreme, a bus for every child. It is also clearly not beneficial or fair to have some students ride a bus for hours on end simply to implement the most inexpensive bus route. An arborescence-style route design would offer effective cost/responsiveness tradeoffs: routes would be inexpensive to implement yet could deliver children to and from school quite quickly.
When planning distribution networks for cable television, companies providing cable service face issues which can be resolved with arborescences. In this model, the root represents the cable distribution hub (which may be a city-wide office, or a neighborhood service hub), and the non-root vertices are the homes of the service consumers. Edges of the service tree represent the physical cable. Clearly, it is expensive for a physical cable to be laid from the distribution hub to every home. Alternatively, sharing cable as much as possible to minimize cable length (and therefore cost), could result in signal degradation caused by excessive cable length. A cable television distribution network could use arborescences as the basis of their delivery service trees. These trees would be inexpensive to implement, yet have low signal degradation.

These are just a few of the many applications of arborescences within service oriented industries. Each requires the low cost and low radius service routes which arborescences can provide.

2.2.2 Multicast Routing

As the use of high bandwidth digital networks and multimedia presentation techniques continue to become an integral part of our economy and society, methods of utilizing these new tools are becoming more and more evident. Applications such as video-conferencing[18], video-on-demand and remote classroom instruction are fast becoming commonplace. These applications all have a voracious appetite for bandwidth. Therefore, it is critical that available bandwidth be used efficiently[12].

The aforementioned applications are all very similar. Each deals with the distribution of data from a single source to multiple destinations. Multicast routing algorithms describe ways of delivering delay-sensitive data from a source to multiple destinations in a network so as to minimize the bandwidth needed to provide the given service to all desired sites.

A common, albeit inefficient and naive, means of providing service to subscribers is called $N$-Unicast. This technique simply sends the same information $N$ times,
once for each current subscriber[12]. With $N$ separate connections and paths being established, it becomes clear that the resulting tree is the graph-based SPT. By consolidating data streams outgoing on a particular link and duplicating them when path divergence becomes necessary, bandwidth is saved, thereby reducing congestion on affected links.

Although there is an equivalent to the SPT in multicast routing, there does not seem to be an equivalent to the MST. This is because global routing information is not easily obtained and changes frequently.

Arborescences can provide a mechanism for efficient multicast routing. The trees generated can provide delay-sensitive service (via their low radius) while using minimal bandwidth (via their low cost).

2.2.3 VLSI Design

Field-Programmable Gate Arrays (FGPAs) provide hardware designers with a fast and inexpensive method of circuit design. Because of their flexibility and reusable components, the VLSI design/validation/simulation cycle can be performed more quickly when using FPGAs than when using more traditional VLSI design techniques. The tradeoff for this flexibility, however, is performance[1]. Inside an FPGA are programmable routing resources. These resources are the secret to the FPGA's flexibility, but it is the internal routing delays these resources incur which limit their performance[1].

Many issues affect these routing delays, and therefore, FPGA performance. First, because FPGA sizes are small and fixed, most designs must be implemented on several FPGAs. It is desirable to implement the design using as few FPGAs as possible, thereby making the design more inexpensive, and allowing a reduction in routing delays between FPGAs. However, the limiting factor in reducing the total number of FPGAs necessary to implement a particular design is often the availability of routing resources. Minimizing routing resources consumption can often result in more compact designs. Routing resource requirements can be minimized using
Steiner routing structures [1].

Second, resource utilization on a typical FPGA does not exceed 80%, so by analyzing critical path characteristics of the design, routing could be optimized along the critical paths using shortest path techniques[1]. A secondary concern is to minimize wirelength to conserve routing resources.

Thus, the tradeoff between shortest path trees and minimum spanning trees each manifest themselves in FPGA design as performance issues. Shorter critical paths increase performance, but conserving wirelength increases performance by reducing the number of FPGAs needed to implement a design. This tradeoff can be addressed using arborescence constructions.

2.3 Previous Work

2.3.1 Rectilinear Steiner Arborescences

As seen in Section 2.2.3, arborescences have applications in VLSI design. Almost exclusively, VLSI applications involve 2-dimensional space and the $L_1$ distance metric (where $dist_1(i, j) = (|i_x - j_x| + |i_y - j_y|)$) rather than the $L_2$ distance metric. For these applications the introduction of Steiner points is a natural and useful extension. The result is the rectilinear Steiner arborescence (RSA) problem. The RSA of a point set is a directed arborescence which minimizes $L_1$ tree cost and connects all points to the origin (root). It must be constructed so all edge endpoints are either the origin, a member of the point set, or a Steiner point. Furthermore, if an edge joins point $p$ to point $q$ and $p$ lies on the path from the root to $q$ within the arborescence, then $p_x \leq q_x$ and $p_y \leq q_y$. The RSA usually examines nodes in the first quadrant of the Euclidean plane, with the root located at the origin.

As with most Steiner tree problems, finding exact solutions to this problem seem to require exponential algorithms. Therefore, the presentation of a polynomial-time algorithm for the RSA by Trubin[24] was quite unexpected. A flaw was exposed by Rao et al.[21], who also propose a heuristic algorithm for generating an RSA whose
cost is no more than twice that of the rectilinear Steiner minimal arborescence (RSMA).

The algorithm of Rao et al. produces an arborescence by iteratively combining pairs of points, \( p \) and \( q \), such that \( \min(p_x, q_x) + \min(p_y, q_y) \) is maximized. That is, the point set is processed from the outside progressing towards the origin. Using a sweepline technique (discussed in Section 3.2), this algorithm requires \( O(n \log n) \) time.

2.3.2 Prim-Dijkstra Tradeoffs

Although we are examining geometric structures in the Euclidean plane, MST and SPT definitions have also been applied to the more general realm of graph theory. Under this generalized environment, a graph \( G = (V, E) \) is given, where \( V = \{v_0, v_1, \ldots, v_n\} \) and each edge \( e_{ij} \in E \) has an associated cost, \( d_{ij} \). The shortest path between \( v_0 \) and \( v_i \) in \( G \) is denoted by \( R_i \). The cost of the unique path from \( v_0 \) to \( v_i \) within a particular subtree of \( G \) is denoted by \( l_i \). Clearly, \( R_i \leq l_i \).

Prim’s algorithm[20] produces a MST, \( T^1 \), by using \( v_0 \) as an initial tree and iteratively adding edge \( e_{ij} \) and vertex \( v_i \) such that \( d_{ij} \) is minimized where \( v_j \in T^1 \) and \( v_i \in V - T^1 \).

Dijkstra’s algorithm[9] creates a SPT, \( T^2 \), using \( v_0 \) as an initial tree. An edge \( e_{ij} \) and a vertex \( v_i \) are iteratively chosen and added to \( T^2 \) to minimize \( l_j + d_{ij} \), where \( v_j \in T^2 \) and \( v_i \in V - T^2 \).

Notice each of the algorithms uses a similar technique: they construct a spanning tree from a single vertex by adding an edge that minimizes a particular constraint.
Although the relationship between the two algorithms may not be clear initially, Alpert et al. [2] placed the minimization objectives side-by-side:

\[ d_{ij} \text{ such that } v_j \in T^1, v_i \in V - T^1 \]
\[ l_j + d_{ij} \text{ such that } v_j \in T^2, v_i \in V - T^2 \]

Recognizing their similarity, the authors introduce a parameter, \(0 \leq c \leq 1\), and combine the above two formulae into:

\[(c \cdot l_j) + d_{ij} \text{ such that } v_j \in T, v_i \in V - T\]

When \(c = 0\), the result is the minimization formula of Prim. However, when \(c = 1\) it is equivalent to Dijkstra’s algorithm. By varying the value of \(c\), a continuous tradeoff between the competing objectives of each algorithm produces arborescences in \(O(n^2)\) time.

The authors examine point sets using the \(L_1\) metric. The tree resulting from this algorithm has a radius that can be shown to be within the constant factor \(\frac{1}{c}\) of the SPT distance. That is, \(c \cdot l_i \leq R_i, \forall \text{ vertices } v_i\). Although Prim’s and Dijkstra’s algorithms perform well for all graphs, performance bounds for the resulting tree with respect to the cost of the MST work only in Euclidean space (of any dimension, \(d\)). In Euclidean space the tree is within \(\log n\) times a constant factor (determined by \(d\) and \(c\)) of the MST cost.

Empirical testing shows that this direct approach to MST-SPT tradeoff trees is quite successful compared to other arborescence generating algorithms. During experimentation, the authors compared their trees to those generated by the algorithms of Cong et al. [5] and Khuller et al. [15], and found significant benefit when tested on typical real-world point sets.
2.3.3 LAST Trees

Another graph algorithm which allows a continuous tradeoff between the MST and the SPT is the Light Approximate Shortest-path Tree (LAST) algorithm of Khuller, et al.[15]. By defining the \((\alpha, \beta)\)-tree introduced earlier, relating \(\alpha\) and \(\beta\) to each other, and devising an algorithm to produce trees adhering to the \((\alpha, \beta)\)-tree property, the authors make a significant contribution to arborescence algorithms.

This arborescence algorithm traverses the MST in a depth first fashion, analyzing each point to determine if the root-vertex distance is within the current tree specifications. If a path length is too long, the edges of the shortest path between root and vertex are added to the tree. Extraneous edges are then removed to maintain the required tree structure.

When \(\alpha > 1\) and \(\beta \geq 1 + 2(\alpha - 1)\) their algorithm will always find an \((\alpha, \beta)\)-tree in \(G\) rooted at a vertex \(r\). The algorithm runs in \(O(m + n \log n)\) time. If the MST and SPT tree are provided it can be computed in linear time.

If better trees are desired, problems begin to arise. If \(\alpha > 1\) and \(1 \leq \beta < 1 + 2/(\alpha - 1)\), there always exists a planar graph \(G\) with a vertex \(r\) such that \(G\) does not contain an \((\alpha, \beta)\)-LAST rooted at \(r\)[15]. In fact, under these conditions, finding an \((\alpha, \beta)\)-LAST is shown to be \(NP\)-complete by reduction from 3-SAT[15].

A special case analyzed by Khuller et al.[15] occurs when \(\alpha = 1\). The \((\alpha, \beta)\)-tree generated is a shortest path tree with cost no more than \(\beta\) times the cost of the MST. When given the SPT of a directed or undirected graph rooted at a given vertex, the tree for this special case can be found in linear time.

2.4 Summary

When generating rooted trees, two goals are often in direct conflict: the desire to have a low cost tree and the desire to have a tree with a low radius. This thesis examines arborescences: undirected rooted trees which provide tradeoffs between low cost and low radius. An arborescence has applications in areas as diverse as VLSI design, network topology design and service route planning. Investigations of
arborescences in the past have dealt with metric spaces using the $L_1$ metric as well as the directed, graph theoretic version of the arborescence. This thesis will contrast that previous research by examining the geometric nature of the problem, including geometric techniques for constructing arborescences.
Chapter 3

Arborescence Algorithm Framework

This chapter develops an algorithmic framework for generating arborescences. As such, it does not provide full algorithmic details on how to generate arborescences, but provides a high level examination of the structures and procedures which can be used to produce arborescence algorithms.

3.1 Regions of Influence

Many algorithms in computational geometry use measures of proximity to aid their progress. Many of these algorithms utilize empty region information to determine proximity relationships: if a predefined region between two points is “relatively” empty, then an edge between them is added. Structures such as the Delaunay Triangulation, Relative Neighborhood Graph (RNG) and Gabriel Graph (GG) all have definitions based upon this notion of an empty region[19].

The Delaunay Triangulation (Figure 3.1(a)) uses a circle through three cocircular vertices as the empty region. If an empty circle can be drawn through three vertices, the edges of the triangle formed by those vertices are added to the DT. Figure 3.1(b) shows the empty circles used to construct the DT in (a).

The Gabriel Graph (Figure 3.1(c)) also uses an empty circle to decide which edges to add. To add an edge between vertices $v_i$ and $v_j$, the region consisting of a circle passing through vertices $v_i$ and $v_j$ and having a diameter of $\text{dist}(i,j)$ must be empty. Figure 3.1(d) shows the empty circles used to construct the GG in (c).
The Relative Neighborhood Graph (Figure 3.1(e)) uses an empty region known as the *lune*. The lune for two vertices $v_i$ and $v_j$ is defined by the intersection of two circles: a circle centered at $v_i$ with radius $\text{dist}(i, j)$ and a circle centered at $v_j$ with radius $\text{dist}(i, j)$. Figure 3.1(f) shows the empty lunes used to construct the RNG in (e).
Other algorithms, however, utilize a region of influence (ROI) associated with each vertex rather than each edge. The ROI of a vertex is an area over which that point is dominant. The shape of a ROI varies with the application, although a circle centered at the associated vertex is quite common. As examples, circle-based regions of influence can be used to find the MST and the Voronoi Diagram.

Figure 3.2(a) depicts three stages of MST development. Circular regions of influence begin to grow around each vertex. When two components collide, an edge is added between them (if they are not already in the same connected component). Eventually the algorithm concludes with the MST being the graph generated.

Figure 3.2(b) shows the development of a Voronoi Diagram using ROI techniques. The circular ROI grows uniformly around each vertex. When two regions collide, they establish a boundary. These boundaries become Voronoi edges which, upon algorithm completion, divide the plane into the separate Voronoi regions.

Although most algorithms use circles as the ROI, other shapes for the region may be useful if given the correct context. The use of regions of influence to build an arborescence might progress in a fashion similar to the construction of the MST in Figure 3.2(a). The size of the ROI will often be dependent upon the stage and progress of the algorithm relative to the vertex position. To manage the growth of the regions of influence (and thereby the progress of the algorithm) a modified version of an algorithmic tool known as the sweepline is used.

3.2 The Sweepcircle Technique

The sweepline (or plane sweep) technique is a commonly used tool in computational geometry[17]. Traditionally, a sweepline crosses the plane, maintaining a data structure along its frontier and performing tasks at discrete intervals (events) as it moves. Without loss of generality, assume the sweepline stands perpendicular to, and moves horizontally along, the x-axis[19]. Conceptually, the line starts at \(-\infty\) (an arbitrarily small value) on the x-axis and moves to \(+\infty\) (an arbitrarily large value). Because events are processed in an ordered fashion across the plane,
the $x$-dimension is treated as a dimension of time, with events having a *timestamp* ($x$-coordinate) associated with them[6].

The sweepline has become a common tool used in geometric algorithm design because it provides an efficient and natural ordering to an otherwise unruly object set. In arborescence design, however, the special status and arbitrary positioning of the root makes the sweepline technique difficult to apply. Therefore, a *sweepcircle* is defined to be a circle, centered at a given point (the root), with varying radius.

Figure 3.2: Applications of Regions of Influence
When an algorithm begins, the sweep circle has an arbitrarily large radius (approximately \( \infty \)), allowing it to encompass the entire vertex set. As the algorithm progresses, the radius of the sweep circle decreases, until the radius becomes zero. In a fashion similar to the sweepline technique, a data structure records objects along the sweep circle frontier and processes new events as they are encountered. The radius of the sweep circle will be treated as a dimension of time, recording timestamps for events based on the sweep circle radius when that event occurs.

Although the algorithm is controlled by the sweep circle, discrete events determine the state and progress of the sweep circle.

### 3.3 Events

An event is an occurrence of interest or significance to the algorithm. These events are based upon the positions of the vertices in the point set. In generating arborescences, there are two types of events which are significant:

- **Point Activation** Event — A particular vertex begins to grow its ROI as the sweep circle reaches it.

- **ROI Intersection** Event — A particular ROI intersects with another ROI.

The position of a vertex relative to the sweep circle determines a number of its properties. In particular, a vertex has a number of *states* in which it can be: *inactive*, *active* and *deactivated*. All vertices begin in the inactive state. A vertex which is expanding its ROI and attempting to establish a path to the root is in the active state. Finally, if a vertex has assured itself a path to the root, it is in the deactivated state.

The size and shape of a ROI for a vertex is determined by the state of the vertex. An inactive vertex has a ROI consisting of itself. In other words, the vertex is significant, but it exercises no influence on the surrounding area until it becomes active. Once activated, the ROI definition and position of the sweep circle frontier...
are used to construct a growing ROI for the vertex. Each deactivated vertex will be
defined to have a ROI consisting only of itself.

As the ROI of an active vertex grows, there will inevitably be contact with other
regions of influence. Given the position of two vertices and the root, it will be
necessary to know when and where their regions of influence will intersect. If it is
chosen carefully, the ROI definition should allow a constant time (i.e., \(O(1)\) time)
computation of the intersection timestamp and location. When a ROI intersection
occurs, the respective partial trees associated with each vertex combine, leaving a
single ROI with the ability to extend edges toward the root.

When a vertex activates, future ROI intersections must be determined. With
potentially every pair of vertices in a point set intersecting, any algorithm considering
all intersections would take at least \(O(n^2)\) time. However, if the ROI has been
defined such that only adjacent regions of influence will intersect, it may be possible
to prune the \(O(n^2)\) intersections down to only \(O(n)\). The ordering of the set of
active components on the surface of the sweepcircle allows the pruning of component
intersections which can never happen. First, however, a tool is needed to maintain
what is known about the sweepcircle frontier. This tool is the sweepcircle adjacency
cycle.

### 3.4 Sweepcircle Adjacency Cycles

As the algorithms progress, a sweepcircle adjacency cycle captures information about
regions of influence present on the sweepcircle frontier. As mentioned above, by
examining only intersections with neighboring regions of influence, the number of
potential intersections can be greatly reduced, thus speeding up the algorithm dra­
matically.

At the top of Figure 3.3(a), a sweepcircle around the root and the ROI produced
by vertex \(A\) can be seen. Vertex \(A\) has no adjacent regions of influence, so in the
lower part of the figure, its adjacency cycle appears as a single vertex. Figure 3.3(b)
has two regions of influence on the sweepcircle frontier resulting in vertex \(B\)'s ROI.

25
being adjacent to vertex A’s ROI and vice versa. This information is captured in (b)’s adjacency cycle. Figures 3.3(a) and (b) are special cases of adjacency cycles. Figure 3.3(c) is a more typical adjacency cycle. Vertex A’s ROI is adjacent to the regions of influence for B and C. Vertex B’s ROI is adjacent to the regions of influence for A and C. Vertex C’s ROI is adjacent to the regions of influence for A and B. This results in the adjacency cycle presented in (c).

As was the case for a sweepline, when an event occurs, the sweepcircle adjacency cycle must be updated to reflect any changes to the sweepcircle frontier. For example, when a vertex becomes active, it establishes its ROI on the sweepcircle frontier. Therefore the adjacency cycle must be updated. In Figure 3.3, vertex F is activated between vertices A and D. As a result, vertex F is added to the adjacency cycle between the nodes representing A and D: edges (A, F) and (D, F) are added and edge (A, D) is removed.
3.5 The Algorithm Framework

The algorithm framework for generating an arborescence is quite simple, yet versatile.

The framework is divided into three stages: preprocessing, the main loop and the postprocessing. Preprocessing primes the event queue by adding point activation events and, if necessary, the first ROI intersection event for each vertex. The main loop of the algorithm processes events until the event queue is empty. If a point activation event is encountered, the sweepcircle adjacency cycle is updated and intersection events for this new active vertex are computed. If the current event is a ROI intersection, the intersecting components are merged into one larger component. Some vertices deactivate, leaving a single active (or inactive) vertex to carry on. These deactivated vertices are then removed from the sweepcircle adjacency cycle; this in turn can cause new ROI intersection events.

At a high level, here is the algorithm framework:

<table>
<thead>
<tr>
<th>[Preprocessing]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given a vertex set ( {r, v_1, v_2, \ldots, v_{n-1}} )</td>
</tr>
<tr>
<td>For each non-root vertex</td>
</tr>
<tr>
<td>Mark vertex as inactive</td>
</tr>
<tr>
<td>Add a Point Activation event for this vertex</td>
</tr>
<tr>
<td>Add an initial ROI Intersection event for this vertex</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[Main Loop]</th>
</tr>
</thead>
<tbody>
<tr>
<td>While there are events to process</td>
</tr>
<tr>
<td>Get next event</td>
</tr>
<tr>
<td>Case type of event is</td>
</tr>
<tr>
<td>Point Activation:</td>
</tr>
<tr>
<td>Activate vertex</td>
</tr>
<tr>
<td>Update sweepcircle adjacency cycle</td>
</tr>
<tr>
<td>Add ROI intersection events appropriate for this vertex</td>
</tr>
</tbody>
</table>
ROI Intersection (assuming neither vertex has been deactivated):
Perform component merge technique
Deactivate appropriate vertices
Update sweepcircle adjacency cycle
Add any ROI intersection events caused by vertex deactivation

[Postprocessing]
Output arborescence

Of course, as the name implies, this is simply a framework. Much of the detail specific to an algorithm is contained within independent modules.

3.6 Independent Framework Modules

Using the above framework to create an arborescence algorithm requires a number of choices. These choices are the “muscles” to be placed on the algorithm “skeleton”. As shall be seen later, a good set of modules can yield good quality arborescences but a single poor choice can ruin the algorithm.

3.6.1 Region of Influence Definition

Algorithms created using this framework will differ primarily in the definition of the region of influence. Even though the ROI could take virtually any shape, it is probable that most will not be of much use in generating arborescences. Still, some definitions look promising. Two of these, the circle and the tangent ROI, will be examined in Chapters 4 and 5. Other possible regions of influence will be discussed in Section 7.2.
When designing and defining a region of influence there are several properties and desirable qualities to bear in mind:

- The size of a ROI is defined by the position of its defining vertex relative to the sweepcircle frontier.

- In the Prim-Dijkstra tradeoff algorithm presented in Section 2.3.2 a parameterized algorithms was used to provide a continuous tradeoff between the MST (cost) and the SPT (radius). Similarly, having a parameterized ROI definition which allows a continuous tradeoff between the MST and the SPT would be desirable.

- As the ROI expands toward the root, it should have an increasing willingness to join with (laterally) adjacent regions of influence.

### 3.6.2 Component Merge Techniques

When a ROI intersection event occurs, edges are added to the arborescence. The involved components are now merged, thereby forming a larger component. The merge itself can occur in a number of different ways. Three will be examined here, and other potential merge techniques will be discussed in Section 7.2.

In the *spanning merge* (and therefore a spanning arborescence), only edges between existing vertices may be added. This is in contrast to the addition of Steiner points, which result in Steiner arborescences and are discussed second. In a spanning merge the edge between the two intersecting vertices is added to the arborescence and the vertex furthest from the root becomes deactivated. In the spanning merge examples seen in Figures 3.4(a) and (b), the regions of influence for vertices i and j collide. Edge \((i, j)\) is added to the arborescence and vertex \(j\) deactivates (because \(j\) is further from the root \(r\)). Vertex \(i\)'s state remains unchanged, allowing its ROI to continue to pursue a path to the root.

The second merge technique is quite difference from the spanning merge, and involves the addition of Steiner points. Recall from Section 2.1 that Steiner points
are often used in tree construction to reduce overall cost. Rules for the introduction and placement of these Steiner points can vary.

The Steiner Minimal Tree uses solely what will be called classic Steinerization techniques. The Steiner points in the SMT minimize the local tree for three noncollinear vertices. When one of the angles in the triangle formed by these vertices is at least $120^\circ$, the minimizing point is the vertex of the obtuse angle[13]. Figure 3.5(a) shows such a case ($\angle jir > 120^\circ$). In this case, the minimizing point is vertex $i$.

If, however, all angles in the triangle are less than $120^\circ$ the Steiner point is located at the Toricelli point. Given a triangle $\triangle ijk$ (as in Figure 3.6(a)), the Toricelli point is formed in the following manner. Three equilateral triangles are constructed outside of $\triangle ijk$, using the three edges of $\triangle ijk$ as a side of each. Figure 3.6(b)
shows the equilateral triangles constructed from edges \((i, j), (i, k)\) and \((j, k)\). Circles circumscribing each equilateral triangle are added (Figure 3.6(c)). The point of intersection of these three circles is the Toricelli point, and therefore, the desired Steiner point\[13\]. Figure 3.6(d) depicts the position of the Steiner point for \(\triangle ijk\). It is interesting to note that all the angles between the Toricelli point and the vertices from which it is defined measure \(120^\circ\).

When two regions of influence collide in an arborescence algorithm, the Toricelli point between the two affected vertices and the root can be used as the Steiner point, as in Figure 3.5(b). In this Figure, Steiner point \(k\) would be added to the point set, edges \((i, k)\) and \((j, k)\) would be added to the arborescence. Vertices \(i\) and \(j\) would deactivate, and \(k\) would begin to develop a path to the root.

Figure 3.6: Finding a Toricelli Point

A third component merge alternative is called point of intersection Steinerization. For this merge technique, the point of intersection between two regions of influence is used as the Steiner point. Because ROI intersections can occur between active and inactive vertices, the Steiner point may be located at the inactive vertex. This is the case observed in Figure 3.7(a). The ROI for vertex \(j\) has intersected with the vertex \(i\). The intersection point would then be vertex \(i\), and no Steiner point would be necessary. Therefore, edge \((i, j)\) would be added to the vertex set, and vertex \(i\) would be allowed to develop a path to the root.

If however, the intersection point did not occur at either vertex, a Steiner point would be added to the point set. This Steiner point would be located where the
intersection occurred. Consider Figure 3.7(b): Vertex $k$ is the point of intersection between the regions of influence for vertices $i$ and $j$. Therefore, edges $(i, k)$ and $(j, k)$ would be added to the arborescence. Points $i$ and $j$ would then deactivate themselves, leaving $k$ to construct a path to the root.

![Figure 3.7: Point of Intersection Steinerization Merge](image)

3.6.2.1 Bounding the Radius

Each of the three merge techniques discussed above have strengths and weaknesses. One potential weakness can be particularly easy to remedy: a bound on a SPT radius approximation. Simple calculations performed during edge addition can ensure for each vertex a $k$-approximation to the SPT radius, for $k \geq 1$. That is, the radius of the resulting tree will be no more than $k$ times larger than the SPT radius. To ensure this bound, whenever an edge is to be added to the arborescence being developed, a check can be made to see if the added edge would cause a violation of the $k$-approximation to the SPT radius for any vertex in the two components to be merged. If the edge would cause a radius violation, the ROI intersection event is ignored. An ignored ROI intersection event between vertices $i$ and $j$ will require the addition of new events for the intersection of $i$ and $j$'s other neighbor in the sweepcircle adjacency cycle, and an intersection event for $j$ and $i$'s other neighbor.
To accomplish the $k$-approximation, every vertex has a current distance variable associated with it. This variable indicates the total additional edge cost that can be added from this vertex on the path to the root before a violation of the $k$-approximation occurs. Each vertex $v_i$ has its current distance variable initialized to be $k$ times its shortest path tree distance to the root. Whenever two components merge, it will be shown below that it can be easily determined if the edge addition will cause a radius bound violation. If the edge would cause a radius bound violation, the current ROI intersection event is ignored.

To efficiently test for a radius bound violation, testing of more than one vertex should be avoided. Therefore, let the subtree of vertex $v_i$ in arborescence $A$ be the set of vertices which pass through $v_i$ on their path in $A$ to the root. Set the current distance variable for vertex $v_i$ to hold the minimum distance any vertex in the subtree of $v_i$ can yet travel without violating the $k$-approximation.

In detail, suppose that the current distance variable for vertex $g$ is denoted by $g.CurDist$, and assume a ROI intersection event has occurred between vertices $i$ and $j$. The decision has been made to add edge $(i, j)$ to the arborescence and deactivate vertex $i$, as long as the radius bound will be maintained after this addition. The edge $(i, j)$ will be added to the arborescence only if $\text{dist}(i, j) + \text{dist}(j, r) \leq i.CurDist$. That is, edge $(i, j)$ will be added only if connecting the component from vertex $i$ to vertex $j$ then directly to the root $r$ (the least additional cost for adding edge $(i, j)$) will not cause a violation of the $k$-approximation.

When a merge is successfully performed, the $CurDist$ variable for this unified component may need to be updated. To update it, $j.CurDist = \min(i.CurDist - \text{dist}(i, j), j.CurDist)$.

The addition of a Steiner point will result in similar situations. The addition of Steiner point $k$ for vertices $i$ and $j$ and edges $(i, k)$ and $(j, k)$ will occur only if $i.CurDist - \text{dist}(i, k) - \text{dist}(k, r) \geq 0$ and $j.CurDist - \text{dist}(j, k) - \text{dist}(k, r) \geq 0$. The $CurDist$ variable for this new component should then be set: $k.CurDist = \min(i.CurDist - \text{dist}(i, k), j.CurDist - \text{dist}(j, k))$. 
Using this technique, the radius bound for any arborescence the framework produces will be maintained, regardless of the component merge technique.

3.6.3 Not Using the Sweepcircle Adjacency Cycle

The sweepcircle adjacency cycle discussed earlier has beneficial effects on the speed of an arborescence algorithm. It allows the pruning of the number of potential ROI intersection events down from $O(n^2)$ to $O(n)$. However, there are times when the sweepcircle adjacency cycle cannot be beneficially used. As was noted in Section 3.3, if a ROI definition allows a ROI to extend inside the sweepcircle, this definition could jeopardize the efficiency improvements realized by the sweepcircle adjacency cycle. In such a case, a point activation event will require the determination of ROI intersection events between the newly activated vertex and each remaining active vertex.

An additional problem may exist when a ROI extends inside the sweepcircle frontier. In such a case, two regions of influence which are not adjacent (on the sweepcircle frontier) may intersect. This can lead to regions called pockets. Figure 3.8(a) shows how and where a pocket can occur. As a sweepcircle contracts, a ROI intersection event involving two active vertices occurs. This intersection point can occur inside the sweepcircle. The area above the point of contact between the regions of influence may contain numerous components, as in Figure 3.8(b). These pockets would complicate the arborescence-generating algorithms. Because identifying and fixing pockets can be quite computationally expensive without clear benefit, the component merge techniques presented here ignore them.

As well, if a region of influence can extend inside the sweepcircle frontier, it can intersect with an inactive ROI. Therefore, when a vertex activates, its first intersection with an inactive vertex should already be determined. The time and location of this first intersection could be computed during the preprocessing phase, or when that vertex activates.
3.7 Data Structures

Developing an algorithm using this framework requires a number of data structures to maintain all the relevant information. A list of pertinent vertex information will be necessary, and as mentioned earlier, an event queue and a sweepcircle adjacency cycle will be required. The vertex table can be implemented as an indexed array. The event queue is simply a priority queue sorted by the timestamp field of the event. It can be best implemented as a heap. The sweepcircle adjacency cycle can be implemented as a height-balanced binary tree. The following is a closer examination of each data structure, including the properties and operations associated with each (the computational complexity for each method is located at the end of each entry):

- **Vertex Table**: An indexed table containing vertex information. Each entry is a record containing the following information:

  Fields:

  - **State**: A variable retaining one of the following vertex states: inactive, active and deactivated. These states affect ROI growth for this vertex.
  - **CurDist**: The CurDist for the inactive or active vertex in any component is the remaining edge cost that can be used to reach the root before the SPT radius approximation for a vertex in that component is violated.
• Event Queue: A priority queue sorted by the timestamp field.

Methods:

- Initialize: Setup an empty event queue. \((O(1))\)
- Empty: Returns true if the event queue contains no pending events and false otherwise. \((O(1))\)
- Insert\(\text{event-type},\{v_1, \ldots, v_i\},\text{time}\): Insert an event-type event into the event queue with a timestamp of time. This event involves vertices \({v_1, \ldots, v_i}\). \((O(\log n))\)
- Remove: Returns the record in the event queue with the largest time-stamp. The event record returned consists of the event-type and a set of the vertices involved in the event. \((O(\log n))\)

• Sweepcircle Adjacency Cycle: A list ordered by clockwise location of active components on sweepcircle frontier.

Methods:

- Initialize: Setup an empty adjacency cycle. \((O(1))\)
- Insert\(i\): Insert vertex \(i\) into its appropriate place in the sweepcircle adjacency cycle ordering. \((O(\log n))\)
- Neighbors\(i\): Returns the pair of vertices which are adjacent to vertex \(i\) in the sweepcircle adjacency cycle ordering. \((O(\log n))\)
- Remove\(i\): Remove vertex \(i\) from the sweepcircle adjacency cycle. The neighbors of \(i\) become adjacent to each other. \((O(\log n))\)

These are the most important data structures used in the arborescence algorithm framework. Other data structures may be necessary for collecting the arborescence edges, etc. The data structures presented here, however, encompass the majority of functionality required by the algorithm framework.
3.8 Arborescence Properties

After the framework has constructed an arborescence algorithm, concerns arise about the quality and properties of the algorithm. Like any other algorithm, the computational complexity and space requirements of these algorithms can be determined. However, the purpose of these algorithms is to generate low-cost and low-radius arborescences. Therefore, this section will introduce three metrics by which the quality of a particular algorithm and its arborescences can be judged.

The arborescences generated by this framework and the component merge techniques will harbor another property particular to rooted trees: the *peelable* property. Recall that a rooted tree in which all edges radiate outwards from the root is an arborescence. A peelable tree has its edges oriented such that on the path from any vertex \( v_i \) to the root \( r \), by leaving vertex \( v_a \) and arriving at vertex \( v_b \) it is guaranteed that \( \text{dist}(v_a, r) \geq \text{dist}(v_b, r) \). In other words, traversing an edge in the direction leading to the root must result in the traversee being no further from the root. Because the spanning merge leaves the vertex nearest the root active, it supports the peelable property. The Steiner points added in the classic Steinerization and point of intersection Steinerization all fall closer to the root than either involved vertex, hence the peelable property is maintained.

Each of the choices for the independent framework modules discussed above (Section 3.6) define a unique algorithm. Similarly, the arborescences produced by each algorithm can be quite unique. Thus, the properties of the arborescences can be unique. Because arborescences offer tradeoffs with respect to cost and radius, these properties will become the metrics by which an algorithm and the arborescences it generates will be judged.

The first metric to be examined is that of cost. As discussed earlier (Section 2.1), the MST is the most inexpensive tree using exactly \( n - 1 \) edges. As a result, it seems natural to compare the weight of the arborescences to that of the MST on the same point set. If Arborescence\((r, v_1, v_2, \ldots, v_{n-1})\) represents the arborescence generated by the algorithm under analysis, the cost ratio will be defined as:
\[
cost_{\text{ratio}} = \frac{|\text{Arborescence}(r, v_1, v_2, \ldots, v_{n-1})|}{|\text{MST}(r, v_1, v_2, \ldots, v_{n-1})|}
\]

Quite obviously, for algorithms utilizing the spanning component merge tech­
nique, \( \cost_{\text{ratio}} \geq 1 \) since the arborescence must cost at least as much as the MST. However, when a component merge technique utilizing Steiner points is used, the cost of the arborescence has a lower bound of the associated SMT cost. Thus, because of the Steiner ratio, \( \cost_{\text{ratio}} \geq \frac{\sqrt{3}}{2} \).

The second metric examined is the arborescence radius. Recall from Section 2.1
that the radius of a tree is the most expensive path between the root and any vertex. The SPT has by definition the smallest radius for a given point set, so it is only
logical to compare the arborescence radius to the that of the SPT:

\[
\text{radius}_{\text{ratio}} = \frac{\rad(\text{Arborescence}(r, v_1, v_2, \ldots, v_{n-1}))}{\rad(\text{SPT}(r, v_1, v_2, \ldots, v_{n-1}))}
\]

Because the Euclidean SPT consists of an edge from every non-root vertex to the
root, the component merge technique will not affect the lower bound of the ratio. That is, \( \text{radius}_{\text{ratio}} \geq 1 \) regardless of the arborescence algorithm used.

The third metric is a similar, but stronger version of \( \text{radius}_{\text{ratio}} \). Previous
research compares their tree radius with the SPT tree radius, which the metric \( \text{radius}_{\text{ratio}} \) does. However, another metric can reveal other arborescence character­
istics. Consider the arborescence in Figure 3.9. The radius of this arborescence is
\( \text{ir} \) because the path between \( i \) and \( r \) through the arborescence is longer than any
other path to \( r \). As well, the SPT radius for this set of vertices is also \( \text{ir} \). Therefore,
\( \text{radius}_{\text{ratio}} = \frac{\text{ir}}{\text{ir}} = 1 \). However, consider the radius of vertex \( j \). It has a measured ra-
dius within the arborescence of 2.51 units. But \( \text{jr} = 1.76 \) units when measured. So,
the path between \( j \) and \( r \) through the arborescence has a radius ratio of \( \frac{2.51}{1.76} \approx 1.43 \). The radius of vertex \( j \) is 43% longer than its SPT radius. The third metric will
capture this information, the \textit{maximum vertex radius ratio}.
Given a set of vertices \( \{ r, v_1, v_2, \ldots, v_{n-1} \} \) and an arborescence on those vertices \( A = Arborescence(r, v_1, v_2, \ldots, v_{n-1}) \), the maximum vertex radius ratio will be:

\[
\text{radius}_{\text{max ratio}} = \max_{i=1 \ldots n-1} \left( \frac{\text{dist}_A(i, r)}{\text{dist}(i, r)} \right)
\]

Because the \( \text{radius}_{\text{ratio}} \) will be one of the ratios considered when computing \( \text{radius}_{\text{max ratio}} \), it is guaranteed that \( \text{radius}_{\text{max ratio}} \geq \text{radius}_{\text{ratio}} \). It is interesting to note that the \( k \)-approximation guarantee technique (using the \( \text{CurDist} \) variable) introduced in Section 3.6.2.1 means not only that \( \text{radius}_{\text{ratio}} \leq k \), but also that \( \text{radius}_{\text{max ratio}} \leq k \) because the \( \text{CurDist} \) variable does not allow any vertex to exceed the \( k \)-approximation.

### 3.9 Summary

The arborescence algorithm framework developed in this chapter is a flexible tool for producing arborescence-generating algorithms. By utilizing independent framework modules for defining regions of influence and merging components, many different arborescence algorithms can be created. Once defined, the quality of the algorithm can be determined by examining the \( \text{cost}_{\text{ratio}}, \text{radius}_{\text{ratio}} \) and \( \text{radius}_{\text{max ratio}} \) metrics for the arborescences it produces. The next two chapters will design four algorithms for arborescence generation and examine the properties of these algorithms.
Chapter 4

Circle Arborescence Algorithm

The first algorithm to be examined is called the *circle arborescence algorithm*. As its name suggests, this algorithm make use of circles for the ROI. However, in order to ensure good progress towards the root, the centers of the circles move in the direction of the root as the ROI grows.

A generalization of the Voronoi Diagram (called the *river Voronoi Diagram*) has been introduced by Sugihara[23] by giving the entire plane an ambient motion. All points in the plane drift uniformly in a particular direction. To describe this motion, Sugihara uses analogies to boats (points) in a river (plane). In this Voronoi Diagram generalization, a point $j$ belongs to region $i$ if boat $i$ can reach point $j$ before any other boat. This reduces to the more familiar Voronoi Diagram when the motion is zero. However, as the speed of the river increases, boats have more difficulty moving against the flow of the river. The author defines a variable $\sigma > 0$ such that $\sigma$ is the ratio between river speed and boat speed.

![Diagram](40)

*Figure 4.1: Sigma Parameter Variations*
In Figure 4.1 the impact of $\sigma$ on the shape of the ROI can be seen. In (a), when $\sigma = 0$ (no flow), the ROI reduces to the familiar uniformly expanding circle structure (as seen in Figure 3.2(b)). When $0 < \sigma < 1$, as it is in Figure 4.1(b), it can be seen how the ambient motion hinders boat movement against the current. When $\sigma = 1$ a critical point is reached as river flow prevents any movement forward. Finally, when $\sigma > 1$ (as in Figure 4.1(d)), river flow forces the ROI back despite attempts to remain motionless.

The unidirectional movement of a River Voronoi Diagram is now modified to direct the water flow into the root vertex of the point set. The river analogy now becomes water flowing down the drain of a basin.

By simply varying the $\sigma$ parameter, the basin will allow the production of trees varying in quality from the MST (when $\sigma = 0$) to the SPT (when $\sigma = \infty$). When $\sigma = 0$, the regions of influence are immobile circles, resulting in a MST algorithm similar to that seen in Figure 3.2(a). When $\sigma = \infty$, the centers of the regions of influence reach the root and deactivate before the ROI radius has a chance to grow larger than 0. The result is a set of edges from the root to every non-root vertex (in other words, the SPT). Clearly, $\sigma$ values between these two extremes provide a method of creating trees with inherent tradeoffs between the MST and the SPT.

4.1 Region of Influence Definition

The investigation of this algorithm will take place for $\sigma = 1$ (as in Figure 4.1(c)). Thus, the region of influence for vertex $i$ will be a circle centered at the location where edge $(i,r)$ intersects the sweepcircle. The radius of this circular ROI will be the distance from the center point to $i$. Thus, the ROI for this algorithm forms as shown in Figure 4.2. Shortly after the point activation event, the associated ROI would appear similar to that in Figure 4.2(a). Figure 4.2(c) shows a ROI after colliding with the root, as its vertex prepares to deactivate. Figure 4.3 depicts the overall shape of the ROI with respect to the sweepcircle if it proceeds unhindered to the root. Figure 4.3 may seem redundant at this point, but this structure will be
seen again in Section 5.1 in an unexpected place.

![Figure 4.2: Regions of Influence: Circle Arborescence Algorithm](image)

4.1.1 Intersecting Regions of Influence

Intersection events for the circle arborescence algorithm can take one of two basic forms. The first of these results from the discussion of ROI characteristics in Section 3.6.3. Because this ROI extends into the interior of the sweepcircle, the ROI for an active vertex can intersect with the ROI of an inactive vertex. In Figure 4.4 the ROI for an active vertex collides with (the ROI for) an inactive vertex.

As explained in Section 3.6.3, ROI intersection events such as the one seen in Figure 4.4 can occur only once for a vertex. This is the ROI intersection event which occurred because a ROI grew unhindered by other regions of influence. To determine with which vertex a ROI with this definition will collide and with what timestamp,
the Voronoi Diagram is used. In Figure 4.5(a) a point set and its associated Voronoi Diagram can be seen. Recall that every point within region $i$ of a Voronoi Diagram is closer to point $i$ than any other point in the point set. Further, the edges in a Voronoi Diagram represent those points which are equidistant between two vertices. Still further, the intersection point of three lines represents a point equidistant from three vertices. Therefore, if a ray is shot from a vertex $i$ to the root and it crosses the $(i,j)$ Voronoi edge (as it does in Figure 4.5(a)), then the ROI for vertex $i$ will first intersect vertex $j$ if allowed to grow unhindered.

Figure 4.5: Ray Shooting in the Voronoi Diagram
**Proposition 1**  A ROI intersection event between an active vertex $i$ and an inactive vertex $j$ in the circle arborescence algorithm will have an event timestamp of

$$
\overline{tr} = \overline{ir} - \frac{\overline{ij}}{2 \cdot \cos \angle jir}
$$

**Proof:** Consider Figure 4.5(b), where $\overline{tu}$ is the bisector of $\angle jit$.

1. $\overline{it} = \overline{jt}$
   
   Point $t$ is the center of the ROI and vertices $i$ and $j$ are each radii on that circle.

2. $\triangle ij t$ is an isosceles triangle.
   
   Two of the sides of $\triangle ij t$ are equal (by step 1).

3. $\angle lij t \cong \angle jit \cong \angle jir$
   
   The base angles of an isosceles triangle are congruent.

   $\angle jit$ and $\angle jir$ are the same angle.

4. $\angle lit u \cong \angle lj tu$
   
   $\overline{tu}$ is the bisector of $\angle lit$.

5. $\overline{tu}$ intersects $\overline{ij}$ at $90^\circ$
   
   $\angle jit \cong \angle lij t$ (by step 3).

   $\angle lit u \cong \angle lj tu$ (by step 4).

   The remaining angle in each triangle must be equal.

   And these angles must sum to $180^\circ$ because $\overline{ij}$ is a straight line.

6. $\cos \angle jir = \frac{\overline{ij}/2}{\overline{it}}$
   
   Cosine identity.

7. $\overline{it} = \frac{\overline{ij}/2}{\cos \angle jir}$

8. $\overline{ir} = \overline{it} + \overline{tr}$

9. $\overline{tr} = \overline{ir} - \overline{it}$

10. $\overline{tr} = \overline{ir} - \frac{\overline{ij}/2}{\cos \angle jir}$

   Substitution of step 7 into step 9.

\qed
The other type of ROI intersection event which can occur is between the regions of influence for two active vertices. Figure 4.6(a) shows such an intersection event.

![Diagram](attachment:image.png)

Figure 4.6: ROI Intersection: Case 2

**Proposition 2** A ROI intersection event between two active vertices A and B in the circle arborescence algorithm will have an event timestamp of:

\[
CR = \frac{AR + BR}{2} \left( \sin \frac{\angle ARB}{2} + 1 \right)
\]

**Proof:** Consider Figure 4.6(b).

1. \(CR = DR\)
   - Both are sweepcircle radii.
2. \(\angle CDR = \angle DCR\)
   - By step 1, \(\triangle CDR\) is an isosceles triangle, hence the base angles are congruent.
3. \(AC = AR - CR\)
4. \(BD = BR - DR\)
5. \(BD = BR - CR\)
   - Substitution of step 1 into step 4.
6. \(CD = AC + BD\)
   - Each are the sum of ROI radii.
7. \( \overline{CD} = \overline{AR} + \overline{BR} - 2 \cdot \overline{CR} \)
   
   Substitution of steps 3 and 5 into step 6.

8. \( \sin \left( \frac{\overline{ARB}}{2} \right) = \frac{\overline{CD}/2}{\overline{CR}} \)
   
   Sine identity.

9. \( \sin \left( \frac{\overline{ARB}}{2} \right) = \frac{\overline{AR} + \overline{BR} - 2 \cdot \overline{CR}}{2 \cdot \overline{CR}} \)
   
   Substitution of step 7 into step 8.

10. \( \overline{CR} = \frac{\overline{AR} + \overline{BR}}{2 \sin \left( \frac{\overline{ARB}}{2} + 1 \right)} \)

   Algebraic manipulation of step 9.

□

With the ROI defined for the circle arborescence algorithm, the component merge technique must now be chosen. Two component merge techniques will be examined using this ROI definition. Each will be treated and examined as a separate algorithm.

### 4.2 Circle Spanning Arborescence

Because the addition of Steiner points may not be appropriate in some applications, the first component merge technique examined for the circle ROI will be a spanning merge. However, because a spanning merge technique can cause trees of an arbitrarily large \( \text{radius ratio} \), the radius limiting technique (using the \textit{current distance} variable) as discussed in Section 3.6.2.1 is adopted to provide a \( k \)-approximation to the SPT radius (where \( k = 2 \)).
### 4.2.1 The Algorithm

The following is the pseudo-code for the circle spanning arborescence algorithm.

Given a set, $S$, of $n$ points in the Euclidean plane, including one special point, $r$, also known as the root.

[Preprocessing]

$VD = \text{Voronoi-Diagram}(S)$

Event-Queue.Initialize

For each $i$, where $i \in S - \{r\}$

Set $i.CurDist = 2 \cdot \text{dist}(i, r)$.

Set $i.State = \text{Inactive}$.

Event-Queue.Insert($Point\ Activation, \{i\}, dist(i, r)$)

[Prime event queue with first inactive ROI intersection.]

Shoot a ray from $i$ to $r$, noting the Voronoi edge, $(i, j)$ the ray crosses first.

Event-Queue.Insert($ROI\ Intersection, \{i, j\}, dist(i, r) - \frac{\text{dist}(i, j)}{2 \cos \angle r ij}$)

[Main Loop]

While not Event-Queue.Empty do

Case Event-Queue.Remove of:

Point Activation Event for vertex $i$

[Activate vertex and insert appropriate events.]

Set $i.State = \text{Active}$.

For each $j \in S - \{r\}$ such that $j.State = \text{Active}$

Event-Queue.Insert($ROI\ Intersection, \{i, j\}, \frac{\text{dist}(i, r) + \text{dist}(j, r)}{2}(\sin \frac{\angle r ij}{2} + 1)$)

Event-Queue.Insert($ROI\ Intersection, \{i, r\}, dist(i, r)/2$)

ROI Intersection Event between vertices $i$ and $j$

[Ignore intersection events involving deactivated points.]

If ($i.State \neq \text{Deactivated}$) and ($j.State \neq \text{Deactivated}$)
then

[Determine which vertex should deactivate.]
If \( dist(i, r) < dist(j, r) \)
then
\[
    near = i
\]
\[
    far = j
\]
else
\[
    near = j
\]
\[
    far = i
\]

[Impose SPT approximation bound.]
If \( far.CurDist - dist(far, near) - dist(near, r) \geq 0.0 \)
then

[Add edge and deactivate far vertex]
add edge \((far, near)\) to arborescence
if \( far.CurDist - dist(far, near) < near.CurDist \)
then
\[
    near.CurDist = far.CurDist - dist(far, near)
\]
Set \( far.State = Deactivated \).

[Postprocessing]
Output arborescence.

4.2.2 Algorithm Analysis

4.2.2.1 Time Complexity Analysis

The algorithm framework consists of 3 phases: preprocessing, the main loop, and postprocessing. Each of these phases are independent of the others. Therefore, the most computationally expensive phase will constitute the complexity of this algorithm.

The preprocessing phase begins with the construction of a Voronoi Diagram of \( S \), which requires \( O(n \log n) \) time. The last portion of the preprocessing phase consists
of a loop which executes \( n - 1 \) times. Each loop iteration performs some constant
time computations, and performs two insertions into an ordered event queue. Each
such insertion requires \( O(\log n) \) time, so the overall time complexity of this loop is
\( O(n \log n) \). The preprocessing phase requires \( O(n \log n) \) time.

The main portion of the algorithm consists of a single loop. Inside this loop are
two event processing sections: one for a point activation event and one for a ROI
intersection event. The point activation event first adds a ROI intersection event to
the event queue for each of the potentially \( n \) active regions of influence, requiring
\( O(n \log n) \) total time. The second event is the ROI intersection event, which requires
constant time to process.

The number of iterations performed by the main loop must now be determined.
Since this is a spanning arborescence algorithm, there could be at most \( n \) point
activation events. However, because the ROI for this algorithm extends inside the
sweepcircle, every possible ROI intersection event must be examined, for a total
of \( O(n^2) \) of them. Each of these must be inserted into the event queue, requiring
\( O(\log n) \) time each. The result is that \( O(n^2 \log n) \) time is spent processing point
activation events, and \( O(n^2) \) time processing ROI intersection events. Therefore,
the main loop of the algorithm requires \( O(n^2 \log n) \) time to complete.

The postprocessing portion of the algorithm simply involves the output of the
arborescence generated. Consisting of \( n - 1 \) edges, this step takes \( O(n) \) time.

The main loop in the algorithm constitutes the processing bottleneck, resulting
in a time complexity of \( O(n^2 \log n) \) time for this algorithm.

4.2.2.2 Space Requirements Analysis

The algorithm framework presented in Chapter 3 requires the use of the vertex table
and priority queue data structures (see Section 3.7). The vertex table requires \( O(n) \)
space, but the space requirements of the priority queue can vary with the algorithm.

In the spanning arborescence algorithm presented above, there will be \( O(n) \) point
activation events. These are inserted during the preprocessing phase. During the
main phase, each point activation event can result in \( O(n) \) ROI intersection events.
If all vertices are active at the same time there will be a total of $O(n^2)$ events in the event queue. Each event record in the event queue has a constant size, so the circle spanning arborescence algorithm requires $O(n^2)$ space.

### 4.2.2.3 Algorithm Properties

**Theorem 1** The cost of a circle spanning arborescence on $n$ vertices has a worst case cost ratio with a lower bound of $\Omega(1 + \frac{\pi}{6})$.

**Note:** $f(n) = \Omega(g(n))$, if and only if there exist positive constants $c$ and $n_0$ such that for all $n > n_0$, $|f(n)| \geq c|g(n)|$. Thus, for all values $n$ larger than $n_0$, the value of $f(n)$ is at least as large as $c \cdot g(n)$. This theorem is therefore establishing that the cost ratio bound for the circle spanning arborescence algorithm is at least $1 + \frac{\pi}{6}$, because (as will be shown) such an example has been found.

**Proof:** Consider Figure 4.7(a). Given the root $r$ and a set of $n$ other vertices, each of which is a distance of 1 from the root, let the distance between vertices $a$ and $b$ also be 1. Therefore, all three sides of $\triangle abr$ are of length 1, which implies that all angles inside $\triangle abr$ are $60^\circ$. Next, place vertex $c$ on the bisector of $\angle arb$ such that $cr = 1$. Then place vertex $d$ on the bisector of $\angle arc$ such that $dr = 1$. Repeat until $n$ points are inside $\angle arb$.

![Figure 4.7: Circle Spanning Arborescence With Poor cost ratio](image)

Figure 4.7(b) depicts the MST of the point set displayed in (a). Because $\triangle arb$ is an equilateral triangle, it is known that $\angle arb = \frac{\pi}{3}$ radians. Therefore, the arc across
\( \angle arb \) with a radius from \( r \) of 1 has a length of \( \frac{\pi}{3} \approx 1.0472 \). Additionally, the MST of these points will require a vertex to attach to the root, which will cost 1. Therefore, the cost of the MST on this point set will have an upper bound of \( \frac{\pi}{3} + 1 \approx 2.0472 \). Because \( \overline{ab} = \overline{br} = 1 \), it is known that the cost of the MST on this point set will have a lower bound of 2. Thus, the cost of the MST \( T \) on this point set will be \( 2 \leq T \leq 1 + \frac{\pi}{3} \approx 2.0472 \). To maximize \( \text{cost}_{ratio} \) the MST will be assumed to have the smallest cost possible. That is, the MST is assumed to have a cost of 2.

Figure 4.7(c) is the arborescence generated by the circle spanning arborescence algorithm. By virtue of the layout of the vertex set, the arborescence consists of an edge from vertex \( a \) to every other vertex. To determine the \( \text{cost}_{ratio} \) of this arborescence, its cost must be calculated. The cost of edge \( \overline{ab} \) is easily determined: it was defined to be of length 1. However, the length of the other edges is a little more difficult to precisely determine. However, because this vertex set was defined carefully, it can be done. First, it is known that every vertex is a distance of 1 from the root. Second, it is known that every point added was placed on the bisector of a previously determined angle. This results in the formation of a series of \( n \) triangles: \( \triangle abr, \triangle acr, \triangle adr \), and so forth. Because two sides and the included angle of each of these triangles is known, the Law of Cosines can be applied. The cost of this tree is therefore:

\[
\overline{ar} + \overline{ab} + \sum_{i=1}^{n} \left( 2 - 2 \cos\frac{60^\circ}{2^i} \right) = 1 + 1 + 2n - 2 \sum_{i=1}^{n} \cos\frac{60^\circ}{2^i} = 2(1 + n - \sum \cos\frac{60^\circ}{2^i})
\]

This however, is not easily simplified. An alternative is to bound the sum of the unknown edges. A lower bound for each is easily obtained. Because \( \overline{ab} = 1 \) and \( \overline{bc} \) bisects \( \angle arb \), it is known that \( \overline{ac} \geq \frac{1}{2} \). By a similar argument \( \overline{ad} \geq \frac{1}{4}, \overline{ae} \geq \frac{1}{8}, \) and so forth. Therefore, a lower bound on the tree cost would be:

\[
\overline{ar} + \overline{ab} + \sum_{i=1}^{n} \frac{1}{2^i} = 1 + 1 + 1 - \frac{1}{2^n} = 3 - \frac{1}{2^n}
\]
In fact, an upper bound on the unknown edge lengths is also easily obtained. Notice each of the missing edges must be contained within the arc upon which they lie. As a result, the associated arc length between each vertex and vertex $a$ will constitute an upper bound on the length of the edge from that vertex to $a$. Because $\angle arb = 60^\circ = \frac{\pi}{3}$ radians and because $\overline{ar} = \overline{br} = 1$, the arc length between $a$ and $b$ is $\frac{\pi}{3}$. Because the point set was formed by iteratively bisecting the angle formed by $a$, $r$ and the non-root vertex nearest $a$, the arc between $a$ and $c$ has a length of $\frac{\pi}{6}$, the arc between $a$ and $d$ has a length of $\frac{\pi}{12}$, and so on. Therefore, an upper bound on the tree cost would be:

$$ \overline{ar} + \overline{ab} + \sum_{i=1}^{n} \frac{\pi}{3 \cdot 2^i} = 1 + 1 + \frac{\pi}{3} \sum_{i=1}^{n} \frac{1}{2^i} = 2 + \frac{\pi}{3} \left( 1 - \frac{1}{2^n} \right) $$

The true value of this can be approximated.

$$ 2 + \frac{\pi}{3} \left( 1 - \frac{1}{2^n} \right) \leq 2 + \frac{\pi}{3} = \frac{6 + \pi}{3} \approx 3.0472 $$

Thus, the cost of the tree $T$ (as shown in Figure 4.7(c)) is $3 \leq |T| \leq \frac{6 + \pi}{3}$. Because the arborescence cost is in the numerator of the $cost_{ratio}$ metric, the largest possible tree cost will maximize the metric. Therefore, $|T| = \frac{6 + \pi}{3} \approx 3.0472$ will be assumed. The $cost_{ratio}$ for this tree can now be determined:

$$ cost_{ratio} = \frac{|Arborescence|}{|MST|} = \frac{(6 + \pi)/3}{2} = \frac{6 + \pi}{6} = 1 + \frac{\pi}{6} \approx 1.5236 $$

□

Although the arborescence in Figure 4.7(c) had a $cost_{ratio}$ slightly larger than 1.5, the radius is precisely 2. Can the radius of this tree be reduced, and what effect would this have on the $cost_{ratio}$ metric? To investigate this, consider Figure 4.8(a), which involves the same point set examined in the previous Theorem. Recall that $\overline{ar} = \overline{br} = \overline{ab} = 1$. Therefore, if the SPT radius bound is lowered below two, vertex
$b$ will not join to $a$ as in Figure 4.7(c), but will join to $r$ as in Figure 4.8(a). The tree length is unchanged, as an edge of length 1 is replaced by another edge of length 1. However, the radius bound of the tree can be reduced from $2$ to $1 + \sqrt{2 - \sqrt{3}} \approx 1.5176$ without otherwise affecting the known theoretical bounds of the resulting arborescence.

![Diagram](image)

\[3 \leq \text{Cost} \leq 3.0472\]
\[\text{Radius} = 1 + \sqrt{2 - \sqrt{3}} \approx 1.5176\]

(a)

\[3.25 \leq \text{Cost} \leq 3.5296\]
\[\text{Radius} = 1.2611\]

(b)

Figure 4.8: Effect of Lowering SPT Bound on $\text{cost}_{\text{ratio}}$

By a similar argument, the SPT radius bound can be lowered to approximately 1.26 and have relatively little affect upon the resulting arborescence. Consider the arborescence in Figure 4.8(b). The lowered radius bound causes segment $\overline{ac}$ to be removed from the arborescence, and segment $\overline{cr}$ to be added. This process can be repeated iteratively until the SPT results, but the cost of the tree begins to grow quickly.

Although this lower bound on worst case $\text{cost}_{\text{ratio}}$ performance of this algorithm may not appear to be tight, no worse examples have been found. Visually, the tree shown in Figure 4.7(c) appears to be quite expensive relative to the MST, but this is not the case. Although worse cases may exist, there has been no indication of their existence. The empirical evidence (which will be presented in Chapter 6) lends credibility to this claim, where the worst $\text{cost}_{\text{ratio}}$ value observed was 1.434. Interestingly, the point set and resulting tree were similar to those used in the above proof.
Theorem 2 The radius of a vertex \( v_i \) in the circle spanning arborescence algorithm will be no more than twice the length of that vertices’ radius in the SPT.

Proof: By induction on the number of intermediate vertices \( e \) on the path from vertex \( v_i \) to \( r \) in the arborescence \( A \). Let \( v_i.CurDist \) refer to the \( CurDist \) variable associated with vertex \( v_i \).

Base Case: When \( m = 0 \) there are no intermediate vertices. In other words, vertex \( v_i \) has an edge directly to \( r \). This is clearly less than twice the cost of the SPT edge between \( v_i \) and \( r \).

Inductive Hypothesis: Assume this property holds for \( e = m \) intermediate vertices. Also, assume vertices on the path from \( v_i \) to \( r \) are \( \{v_i, v_{i+1}, \ldots, v_{i+m}, r\} \).

Therefore, to rephrase this property notationally, assume that \( dist_A(v_i, v_{i+m}) + v_{i+m}.CurDist \leq 2 \cdot dist(v_i, r) \).

Inductive Step: Show this property holds for \( e = m + 1 \) intermediate vertices.

Referring to the path from \( v_i \) to \( r \) in \( A \), if vertex \( v_{i+m+1} \) is added to the path between \( v_{i+m} \) and \( r \), then the following must be true: \( dist(v_{i+m}, v_{i+m+1}) + dist(v_{i+m+1}, r) \leq v_{i+m}.CurDist \) (otherwise \( v_{i+m+1} \) would not be added to the path). Thus, it is known that

\[
dist_A(v_i, v_{i+m}) + v_{i+m}.CurDist \leq 2 \cdot dist(v_i, r)
\]

and

\[
dist(v_{i+m}, v_{i+m+1}) + dist(v_{i+m+1}, r) \leq v_{i+m}.CurDist
\]

Thus,

\[
dist_A(v_i, v_{i+m}) + v_{i+m}.CurDist + dist(v_{i+m}, v_{i+m+1}) + dist(v_{i+m+1}, r) \leq 2 \cdot dist(v_i, r) + v_{i+m}.CurDist
\]

which can be simplified

\[
dist_A(v_i, v_{i+m}) + dist(v_{i+m}, v_{i+m+1}) + dist(v_{i+m+1}, r) \leq 2 \cdot dist(v_i, r)
\]

\[
dist_A(v_i, v_{i+m+1}) + dist(v_{i+m+1}, r) \leq 2 \cdot dist(v_i, r)
\]

54
Therefore, the property holds for \( e = m + 1 \) intermediate vertices.

\( \Box \)

**Theorem 3** The radius of the arborescence generated by the circle spanning arborescence algorithm is a 2-approximation of the SPT radius.

**Proof:** By Theorem 2, every vertex in an arborescence using the radius bounding technique will have a radius which is no more than twice that of the SPT radius. Since the radius of a tree is determined by the maximum radius of any vertex in the tree, the arborescence radius will be a 2-approximation to the SPT radius. \( \Box \)

### 4.3 Steiner Circle Arborescence

The use of Steiner points in tree creation can reduce the size of the resulting tree significantly. Therefore, our second algorithm using the circular ROI will use the classic Steinerization merge technique. The use of the artificial radius bounding techniques discussed in Section 3.6.2.1 will be unnecessary, as shall be seen shortly (Section 4.3.2.3).

#### 4.3.1 The Algorithm

The following is the pseudo-code for the circle Steiner arborescence algorithm.

```plaintext
Given a set, \( S \), of \( n \) points in the Euclidean plane, including one
special point, \( r \), also known as the root.

[Preprocessing]
VD = Voronoi-Diagram(\( S \))
Event-Queue.Initialize
For each \( i \), where \( i \in S - \{r\} \)
    Set \( i.\text{State} = \text{Inactive} \).
    Event-Queue.Insert(Point Activation,\( \{i\} \), \( \text{dist}(i, r) \))
```

55
Shoot a ray from $i$ to $r$, noting the Voronoi edge, $(i, j)$ the ray crosses first.

$\text{Event-Queue.Insert}(\text{ROI Intersection}, \{i, j\}, \text{dist}(i, r) - \frac{\text{dist}(i, j)}{2 \cos \angle r ij})$

[Main Loop]

While not Event-Queue.Empty do

Case Event-Queue.Remove of:

- **Point Activation** Event for vertex $i$
  
  Set $i.\text{State} = \text{Active}$
  
  For each $j \in S - \{r\}$ such that $j.\text{State} = \text{Active}$
  
  $\text{Event-Queue.Insert}(\text{ROI Intersection}, \{i, j\}, \frac{\text{dist}(i, r) + \text{dist}(j, r)}{2(\sin \frac{\angle r ij}{2} + 1)})$

  $\text{Event-Queue.Insert}(\text{ROI Intersection}, \{i, r\}, \frac{\text{dist}(i, r)}{2})$

- **ROI Intersection** Event between vertices $i$ and $j$

  If $(i.\text{State} \neq \text{Deactivated})$ and $(j.\text{State} \neq \text{Deactivated})$

  [Ignore intersections involving deactivated points]

  then

  If Steinerization of $i$, $j$ and $r$ involves adding a Steiner point, $k$

  then [All angles in $\triangle ij r < 120^\circ$]

  Add entry for vertex $k$ in Vertex Table.

  Set $k.\text{State} = \text{Inactive}$

  Set $i.\text{State} = \text{Deactivated}$

  Set $j.\text{State} = \text{Deactivated}$

  Add edges $(i, k)$ and $(j, k)$ to arborescence.

  $\text{Event-Queue.Insert}(\text{Point Activation}, \{k\}, \text{dist}(k, r))$

  $\text{TEMP} = S \cup \{k\}$

  $S = \text{TEMP}$

  $\text{VD} = \text{Voronoi-Diagram}(S)$
For each $u$, where $u \in S - \{r\}$

Shoot a ray from $u$ to $r$, noting the Voronoi edge, $(u, v)$ the ray crosses first.

If $(u = k)$ or $(v = k)$

then

$$\text{Event-Queue.Insert}(ROI\ Intersection, \{u, v\}, \text{dist}(u, r) - \left(\frac{\text{dist}(u, v)}{2 \cdot \cos \angle ruv}\right))$$

[no Steiner point necessary]

If $\angle lirj \geq 120^\circ$

then

Add edge $(i, r)$ to arborescence.
Add edge $(j, r)$ to arborescence.
Set $i.\text{State} = \text{Deactivated}$
Set $j.\text{State} = \text{Deactivated}$

else

Add edge $(i, j)$ to arborescence.
If $\text{dist}(i, r) < \text{dist}(j, r)$

then

Set $j.\text{State} = \text{Deactivated}$

else

Set $i.\text{State} = \text{Deactivated}$

[Postprocessing]

Output arborescence.

### 4.3.2 Algorithm Analysis

#### 4.3.2.1 Time Complexity Analysis

The time complexity analysis for the circle Steiner arborescence algorithm will be very similar to the analysis for the circle spanning arborescence algorithm. This
algorithm makes use of Steiner points if they are deemed beneficial. However, there can be at most \( n - 2 \) Steiner points in a tree on \( n \) vertices. Therefore, there are still \( O(n) \) vertices to consider.

As was the case for the circle spanning arborescence algorithm, the most computationally expensive of the three phases must be determined. The preprocessing and postprocessing portions of this algorithm remain unchanged in comparison to the circle spanning arborescence algorithm, requiring \( O(n \log n) \) and \( O(n) \) time, respectively. As well, the general structure for the main loop is identical in both. The difference between the algorithms occurs when processing ROI intersection events.

When processing a ROI intersection event, decisions are made about whether to add a Steiner point to the vertex set. The addition of a Steiner point requires the insertion of a point activation event, an update of the Voronoi Diagram, and the insertion of an ROI intersection event. This will require \( O(n \log n) \) total time per Steiner point. Given that there can be only \( O(n) \) Steiner points, it will take \( O(n^2 \log n) \) time to process all Steiner points. If a Steiner point is unnecessary, the ROI intersection event requires constant time to process. The complexity of point activation events carry forward from the circle spanning arborescence algorithm, requiring \( O(n^2 \log n) \) time to process them all.

The time requirements for the main loop of the circle Steiner arborescence is \( O(n^2 \log n) \) time. Because the main loop is the most computationally expensive portion of the algorithm, the circle Steiner arborescence algorithm requires \( O(n^2 \log n) \) time.

4.3.2.2 Space Requirements Analysis

As was the case for the circle spanning arborescence algorithm, the space requirements for the event queue must be determined.

In the circle Steiner arborescence algorithm presented above, there will be \( O(n) \) point activation events. These are inserted during the preprocessing phase. During the main phase, each point activation event can result in \( O(n) \) ROI intersection events. If all vertices are active at the same time, there will be a total of \( O(n^2) \)
events in the event queue. The addition of $O(n)$ Steiner points results in $O(n)$ vertices in the point set, so there are still $O(n^2)$ events in the event queue.

Each event record in the event queue has a constant size, so the circle Steiner arborescence algorithm requires $O(n^2)$ space.

4.3.2.3 Algorithm Properties

**Theorem 4** The cost of a circle Steiner arborescence on $n$ vertices has a worst case cost ratio with a lower bound of $\Omega(\log n)$.

**Proof:** Consider Figure 4.9. Given $n = 2^i$ evenly spaced points on a circular arc of angle $\theta = 1$ radian and which are a distance of 1 from the root, the cost of the MST would be: $r + r\theta = 1 + 1 \cdot 1 = 2$.

Now consider the cost of the circle Steiner arborescence on the same point set. The points in the circle Steiner arborescence lie at a number of levels. Level $i$ consists of $2^i$ points at distance $r_i = r = 1$ from the root. Level $i - 1$ consists of $2^{i-1}$ points at a smaller distance $r_{i-1}$ from the root, and so on.

![Figure 4.9: Circle Steiner Arborescence With Poor cost\textsubscript{ratio}](image)

Let the cost of level $j$ of the circle Steiner arborescence be $T_j$. This cost is greater than $\frac{1}{2}$ the arc length at level $j$. Therefore, $T_j \geq \frac{\theta r_j}{2}$, where $r_j$ is the radius
at level \( j \). Note that the change in radius, \( \Delta r_j = r_j - r_{j-1} \) decreases as the level number increases. In other words, levels get closer to each other at the higher levels. Therefore, \( r_j \geq \frac{j}{i} \cdot r_i \) since the right hand side of this inequality assumes that the radius \( r_j \) at level \( j \) is smaller than it actually is. Therefore, \( T_j \geq \frac{\theta}{2} \cdot r_i \).

The value of \( T = \sum_i T_j \) must now be determined:

\[
T = \sum_i T_j \geq \sum_{j=1}^{i} \frac{\theta}{2} \cdot r_i \cdot \frac{j}{i} = \frac{\theta r_i}{2i} \sum_{j=1}^{i} j = \frac{\theta r_i}{2i} \left( \frac{i(i+1)}{2} \right) = \frac{\theta r_i i + 1}{2} = \Omega(i) = \Omega(\log n)
\]

Therefore, an arborescence of this configuration will have a \( \text{cost}_{\text{ratio}} \) of \( \Omega(\log n) \). The worst case \( \text{cost}_{\text{ratio}} \) for the circle Steiner arborescence algorithm is at least \( \Omega(\log n) \).

\[\Box\]

**Theorem 5** The radius of a vertex \( v_i \) in the circle Steiner arborescence algorithm will be no more than twice the length of \( v_i \)'s radius in the SPT.

**Proof:** Classic Steinerization techniques dictate that to minimize the tree between three vertices all angles between edges in the resulting tree must be at least \( 120^\circ \).

Given that edge \( (i,j) \) will be in the arborescence, where will \( i \) be relative to \( j \) to maximize the path from \( i \) to the root in the arborescence?

![Diagram](image)

**Figure 4.10:** Maximizing a Local Tree

It is known that \( i \) will lie somewhere beyond \( j \) from \( r \) in a \( 120^\circ \) cone emanating from \( j \). The shaded region in Figure 4.10(a) depicts this region. By fixing \( ir, i_j \)
is maximized (and hence the path from \( i \) to \( r \) inside this tree is maximized) when 
\[ \angle rji = 120^\circ. \]
Because this leads to the longest possible tree between \( i, j \) and \( r \) in 
this cone, it will be assumed without loss of generality that \( i \) will be on the 
120° cone boundary.

It will now be shown by induction on the number of intermediate vertices between 
\( i \) and the root, that the path from \( i \) to \( r \) in arborescence \( A \) will be no more than 
\[ 2 \cdot \overline{ir}. \]

**Base Case:** Assume there is one vertex on the path from \( i \) to \( r \) in the arborescence. 
Consider Figure 4.10(b). Because \( \angle ijr = 120^\circ \), edge \((i, r)\) is the longest edge in \( \triangle ijr \).
Therefore, \( \overline{ij} \leq \overline{ir} \) and \( \overline{jr} \leq \overline{ir} \). Thus, \( \overline{ij} + \overline{jr} \leq 2 \cdot \overline{ir}. \)

**Inductive Hypothesis:** Assume the property holds when there are \( n \) vertices 
between \( i \) and \( r \) on the path inside the arborescence.

**Inductive Step:** Show the property holds when there are \( n + 1 \) vertices between 
\( i \) and \( r \). Consider Figure 4.10(c). Without loss of generality we will assume 
\( \overline{jr} = 1 \). Therefore, by the inductive hypothesis the distance between \( j \) and \( r \) in 
\( A \) will be at most 2. It needs to be shown that \( \text{dist}_A(j, r) + \text{dist}(i, j) \leq 2 \cdot \overline{ir} \), or 
\[ \frac{\text{dist}_A(j, r) + \text{dist}(i, j)}{2 \cdot \overline{ir}} \leq 1. \]
When is \( \rho = \frac{\text{dist}_A(j, r) + \text{dist}(i, j)}{2 \cdot \overline{ir}} \) maximized?

Before \( \rho \) is maximized, it can be simplified. By the Law of Sines, it is known 
that \( \overline{ir} = \frac{\overline{jr} \cdot \sin 120^\circ}{\sin \theta} = \frac{\sqrt{3}}{2 \cdot \sin \theta}. \) Similarly, \( \overline{ij} = \frac{\overline{jr} \cdot \sin(60^\circ - \theta)}{\sin \theta}. \) Recall that by the inductive 
hypothesis, \( \text{dist}_A(j, r) \leq 2 \cdot \overline{jr} = 2. \) Substituting these values into \( \rho \) the following is 
obtained: 
\[ \rho \leq \frac{2 + (\sin(60^\circ - \theta)/\sin \theta)}{(2\sqrt{3}/(2 \cdot \sin \theta))} = \frac{2 \cdot \sin \theta + \sin(60^\circ - \theta)}{\sqrt{3}}. \]

To find the critical point (and potential maximum) for \( \rho \) find 
\[ \frac{d}{d\theta} \rho: \frac{d}{d\theta} \left( \frac{1}{\sqrt{3}} \cdot \left( 2 \cdot \sin \theta + \sin(60^\circ - \theta) \right) \right) = \frac{1}{\sqrt{3}} \cdot \left( 2 \cdot \cos \theta - \cos(60^\circ - \theta) \right). \] This derivative is set to zero and 
solve for \( \theta \):

\[ 0 = \frac{1}{\sqrt{3}} \cdot \left( 2 \cdot \cos \theta - \cos(60^\circ - \theta) \right) \]
\[ \cos(60^\circ - \theta) = 2 \cdot \cos \theta \]
\[ \cos 60^\circ \cdot \cos \theta + \sin 60^\circ \cdot \sin \theta = 2 \cdot \cos \theta \]
\[ \frac{\cos \theta}{2} + \frac{\sqrt{3} \cdot \sin \theta}{2} = 2 \cdot \cos \theta \]
\[ \cos \theta + \sqrt{3} \cdot \sin \theta = 4 \cdot \cos \theta \]

61
\[
\sqrt{3} \sin \theta = 3 \cos \theta \\
\frac{\sqrt{3} \sin \theta}{3 \cos \theta} = 1 \\
\frac{1}{\sqrt{3}} \cdot \tan \theta = 1 \\
tan \theta = \sqrt{3} \\
\theta = 60^\circ
\]

Substituting \( \theta \) into the expressions for \( \overline{i r} \) and \( \overline{i j} \), \( \rho \) reaches a critical point when \( \theta = 60^\circ \). By taking the second derivative of \( \rho \) and setting it equal 0, it is easy to verify that this critical point is indeed a maximum. Therefore, \( \rho \) is maximized when \( \overline{i r} = 1 \) and \( \overline{i j} = 0 \).

Because \( \rho \) is maximized when \( \overline{j r} = \overline{i r} = 1 \), \( dist_A(i, r) \leq 2 \cdot dist(i, r) \).

**Theorem 6** The radius of the arborescence generated by the circle Steiner arborescence algorithm will have a radius which is no more than twice as large as the SPT radius of the same vertex set.

**Proof:** According to Theorem 5, the radius of any vertex in a circle Steiner arborescence is no more than twice its corresponding SPT radius. Because the radius of a tree is the maximum of its vertex's radii, the radius of this arborescence will be no more than twice the SPT radius on the same point set.

### 4.4 Summary

Two arborescence algorithms are presented in this chapter. Each use a circle centered on the sweepcircle as the ROI. The first algorithm, the circle spanning arborescence algorithm, uses a spanning merge technique to join components when a ROI intersection occurs. The second algorithm, the circle Steiner arborescence algorithm, uses a classic Steinerization merge technique to join components. These algorithms each have a computational complexity of \( O(n^2 \log n) \), and require \( O(n^2) \) space.
Chapter 5

Tangent Arborescence Algorithm

The second and final algorithm to be examined is called the tangent arborescence algorithm. The ROI definition for this algorithm is much different than the circle arborescence ROI.

5.1 Region of Influence Definition

The circle arborescence algorithm in the previous chapter has some nice properties. However, its drawbacks include its time complexity \(O(n^2 \log n)\). This complexity is primarily due to the ROI, which extends inside the sweepcircle, requiring more brute force techniques to find ROI intersection events. The examination of a ROI which does not extend inside the sweepcircle would be interesting: it should be more efficient (depending on the ROI). But how would the arborescences generated compare?

The region of influence for the tangent arborescence algorithm is based upon a visibility property. As a sweepcircle collapses, vertices begin to become “visible” to each other over the horizon of the sweepcircle. This occurs when the line between two vertices is tangent to the sweepcircle. In Figure 5.1 such an occurrence can be seen. This structure becomes the basis for the next ROI.
The ROI consists of the intersection of three regions:

- \( \tau_1 \) — The exterior of the sweepcircle. (Example: Figure 5.2(a))

- \( \tau_2 \) — The interior of the angle formed by the region's defining vertex and two lines tangent to the sweepcircle. (Example: Figure 5.2(b))

- \( \tau_3 \) — The interior of the angle formed by the points of tangency for \( \tau_2 \) and using the root as the apex of the angle. (Example: Figure 5.2(c))

![Figure 5.2: Defining a ROI for the Tangent Arborescence Algorithm](image)

Thus, the arborescence ROI is defined by \( \tau_1 \cap \tau_2 \cap \tau_3 \) (Example: Figure 5.2(d)).

As discussed in Section 3.6.3, utilizing the sweepcircle in conjunction with a ROI which does not extend inside the sweepcircle can be of significant benefit from a computational complexity perspective. Therefore, the decision was made to examine the tangent arborescence as described above. In some respects it seems like a reasonable ROI choice. Consider three vertices all the same distance from the root. The visibility property of this ROI ensures that the vertices nearest each other will intersect first. However, the arborescences produced have some shortcomings as shall be seen later. Modifications will be made which will result in arborescences with more desirable qualities.
In Figures 5.3 the change experienced by an arborescence ROI as the sweepcircle collapses can be seen. By the time the ROI meets with the root (as in Figure 5.3(d)) the ROI has affected a larger area than one might expect. In Figure 5.4 the overall area this ROI has influenced can be seen. Its similarity to the ROI shape in Figure 4.3 was an unexpected, but pleasant surprise.

![Figure 5.3: Region of Influence: Tangent Arborescence Algorithm](image)

![Figure 5.4: ROI Overall Shape: Tangent Arborescence Algorithm](image)

5.1.1 Intersecting Regions of Influence

Intersection events for the circle arborescence algorithm can take one of two basic forms. The first is the most typical: the regions of influence for two vertices intersect when they become “visible” to each other. In Figure 5.5 depicts just such an occurrence.
Proposition 3 A ROI intersection event between two regions of influence (defined by points A and B), neither of which are on the sweepcircle frontier, has an event timestamp of

\[ CR = \cos(90^\circ - \angle BAR) \cdot AR \]

Proof: Consider Figure 5.5(b).

1. \( \overline{AB} \perp \overline{CR} \)
   Line segment \( \overline{AB} \) is tangent to the sweepcircle at point \( C \).
2. \( \cos \angle ARC = \frac{CR}{AR} \)
   Cosine identity.
3. \( \angle ARC = 180^\circ - \angle ACR - \angle CAR \)
   The interior angles of a triangle sum to \( 180^\circ \).
4. \( \angle ARC = 90^\circ - \angle BAR \)
   Simplification of step 3 using step 1.
   The angles \( \angle BAR \) and \( \angle CAR \) are the same.
5. \( \frac{CR}{AR} = \cos(90^\circ - \angle BAR) \)
   Transitivity between steps 2 and 4.
6. \( CR = \cos(90^\circ - \angle BAR) \cdot AR \)
   Algebraic manipulation of step 5.
The second type of ROI intersection event takes place when a vertex activates itself inside the ROI of another active vertex. The example in Figure 5.6 shows this situation. Clearly the timestamp for this type of ROI intersection event will be \( BR \), since the intersection event occurs when vertex \( B \) activates.

![Figure 5.6: ROI Intersection: Case 2](image)

With the ROI defined for the tangent arborescence algorithm, the component merge technique must now be chosen. Four component merge techniques were considered, two of which were dismissed and two of which will be examined in detail.

### 5.2 Spanning Tangent Arborescences

As was the case for the circle arborescence, the first merge type considered was the spanning merge. The nature of the ROI for the tangent arborescence tends to have ROI intersections involving vertices, neither of which are on the sweepcircle (like those seen in Figure 5.5. As such, the progress towards the root can be extremely limited.

Consider the spanning tangent arborescence in Figure 5.7. The addition of each edge increases the arborescence cost dramatically, yet yields virtually no progress toward the root. In fact, such a tree would have a cost and radius which is an unbounded factor larger than the MST and SPT, respectively.

The radius bounding technique introduced in Section 3.6.2.1 could limit the arborescence radius, but the arborescence cost would probably remain unacceptable.
Therefore, the potential of the spanning tangent arborescence seems quite limited.

5.3 Unrestricted Tangent Arborescence

The unrestricted tangent arborescence algorithm is the first tangent arborescence algorithm to be fully examined. The purpose of the term “unrestricted” will be become clear in Section 5.5, where a modified (i.e., “restricted”) version of the ROI definition is used.

The apparent poor quality of spanning tangent arborescences leaves Steinerization techniques as the available merging methods. The unrestricted tangent arborescence will use the classic Steinerization merge technique.

5.3.1 The Algorithm

The following is the pseudo-code for the unrestricted tangent arborescence algorithm.

Given a set, S, of n points in the Euclidean plane, including one special point, r, also known as the root.

[Preprocessing]

Sweepcircle-Adjacency.Initialize.

Event-Queue.Initialize.

For each $i$, where $i \in S - \{r\}$

Set $i.State = Inactive.$

Event-Queue.Insert($Point Activation, \{i\}, dist(i, r)$)

[Main Loop]

While not Event-Queue.Empty do

Case Event-Queue.Remove of:

$Point Activation$ Event for vertex $i$

Set $i.State = Active$

Sweepcircle-Adjacency.Insert($i$)

For each $j \in$ Sweepcircle-Adjacency.Neighbors($i$)

If $\angle jir \geq 90^\circ$

then

[i has activated inside another ROI]

Event-Queue.Insert($ROI Intersection, \{i, j\}, dist(i, r)$)

else

Event-Queue.Insert($ROI Intersection, \{i, j\}$,

$\cos(90^\circ - \angle rij) \cdot dist(i, r)$)

Event-Queue.Insert($ROI Intersection, \{i, r\}, 0.0$)

$ROI Intersection$ Event between vertices $i$ and $j$

If ($i.State \neq Deactivated$) and ($j.State \neq Deactivated$)

[Ignore intersections involving deactivated points.]

then

If Steinerization of $i, j$ and $r$ involves adding a Steiner point, $k$

then [All angles in $\triangle ijr < 120^\circ$]

Add entry for vertex $k$ in Vertex Table.

Set $k.State = Inactive.$

Set $i.State = Deactivated.$

Set $j.State = Deactivated.$
Add edges \((i, k)\) and \((j, k)\) to arborescence.

Event-Queue.Insert\((Point\ Activation, \{k\}, dist(k, r))\)

\[m_1 = \text{Sweepcircle-Adjacency.Neighbors}(i) \text{ such that } j \notin m_1\]

\[m_2 = \text{Sweepcircle-Adjacency.Neighbors}(j) \text{ such that } i \notin m_2\]

Event-Queue.Insert\((ROI\ Intersection, m_1 \cup m_2, \cos(90° - \angle m_1 m_2) \cdot dist(m_1, r))\)

Sweepcircle-Adjacency.Remove\((i)\)

Sweepcircle-Adjacency.Remove\((j)\)

else

[no Steiner point necessary]

If \(\angle irj \geq 120\)

then

\([i \text{ and } j \text{ are on opposing sides of } r]\)

Add edge \((i, r)\) to arborescence.

Add edge \((j, r)\) to arborescence.

Set \(i.\text{State} = \text{Deactivated}\)

Set \(j.\text{State} = \text{Deactivated}\)

\[m_1 = \text{Sweepcircle-Adjacency.Neighbors}(i) \text{ such that } j \notin m_1\]

\[m_2 = \text{Sweepcircle-Adjacency.Neighbors}(j) \text{ such that } i \notin m_2\]

Event-Queue.Insert\((ROI\ Intersection, m_1 \cup m_2, \cos(90° - \angle m_1 m_2) \cdot dist(m_1, r))\)

Sweepcircle-Adjacency.Remove\((i)\)

Sweepcircle-Adjacency.Remove\((j)\)

else

\([i \text{ and } j \text{ are on same side of } r]\)

Add edge \((i, j)\) to arborescence.

If \(dist(i, r) < dist(j, r)\)

then

Set \(j.\text{State} = \text{Deactivated}\).
Given \( \{m, n\} \in \text{Sweepcircle-Adjacency.Neighbors}(j) \)

\[
\text{Event-Queue.Insert(ROI Intersection,} \{m, n\}, \cos(90° - \angle rmn) \cdot \text{dist}(m, r))
\]

\( \text{Sweepcircle-Adjacency.Remove}(j) \)

else

\( \text{Set } i.\text{State} = \text{Deactivated}. \)

Given \( \{m, n\} \in \text{Sweepcircle-Adjacency.Neighbors}(i) \)

\[
\text{Event-Queue.Insert(ROI Intersection,} \{m, n\}, \cos(90° - \angle rmn) \cdot \text{dist}(m, r))
\]

\( \text{Sweepcircle-Adjacency.Remove}(i) \)

[Postprocessing]

Output arborescence.

### 5.3.2 Algorithm Analysis

#### 5.3.2.1 Time Complexity Analysis

As outlined in the arborescence algorithm framework, the Tangent Arborescence algorithm is divided into three distinct sections. The preprocessing section of this algorithm is somewhat simplified compared to its counterpart in the Circle Arborescence algorithm. However, the preprocessing complexity remains unchanged, requiring \( O(n \log n) \) time. As well, the postprocessing portion is identical, requiring \( O(n) \) time.

In the main loop of the algorithm there are once again two types of events to process: point activation events and ROI intersection events. The point activation events are slightly less complex than those previously seen. Each point activation creates at most three more events and performs some manipulations of the sweep-circle adjacency cycle. Each point activation event requires \( O(\log n) \) time. There are still \( n - 1 \) initial point activation events (and another \( n - 2 \) for Steiner points), thus requiring \( O(n \log n) \) total time.
Additionally, ROI intersection events in the Tangent Arborescence are quite straightforward. If a Steiner point is deemed beneficial, at most two events are generated, the sweepcircle adjacency cycle is updated twice and queried twice, requiring $O(\log n)$ time for each Steiner point. If a Steiner point is not added, then one event is generated and at most two queries and two updates are made to the sweepcircle adjacency cycle, requiring $O(\log n)$ total time. Because there can be only $O(n)$ ROI intersection events, they require $O(n \log n)$ total time.

Each portion of the main loop requires $O(n \log n)$ time, so the restricted tangent arborescence algorithm requires $O(n \log n)$ total time.

### 5.3.2.2 Space Requirements Analysis

The algorithmic framework requires at least $O(n)$ space simply to store vertex information and the resulting arborescence.

Each vertex (including Steiner points) will require a point activation event. This results in $O(n)$ events in the queue. Each time a vertex activates and/or deactivates there are a constant number of events added to the event queue. Since each vertex must activate and deactivate, there will be $O(n)$ events in the event queue. Each event requires a constant amount of space.

The sweepcircle adjacency cycle may contain only $O(n)$ vertices, each of which require a constant amount of space, so the space requirements of the restricted tangent arborescence algorithm is $O(n)$.

### 5.3.2.3 Algorithm Properties

**Theorem 7** *The cost of an unrestricted tangent arborescence on n vertices has a worst case cost ratio with a lower bound of $\Omega(\log n)$.*

**Proof:** The proof is identical to that used to show Theorem 4 from Section 4.3.2.3. In each case, the cocircular points merge with their respective neighbors. Each use the same merge technique (classic Steinerization) ensuring the progress of the algorithm will be identical for these point sets. □
**Theorem 8** The radius of a vertex \( v_i \) in the unrestricted tangent arborescence algorithm will be no more than twice the length of that vertex's radius in the SPT.

**Proof:** The proof is identical to that used to show Theorem 5 from Section 4.3.2.3. Each use the classic Steinerization technique, which always uses the root as one of the three vertices used to locate the Toricelli point. Therefore, the paths generated always progress toward the root, at the same rate as in a circle Steiner arborescence. \( \Box \)

**Theorem 9** The radius of the arborescence generated by the circle Steiner arborescence algorithm will have a radius which is no more than twice as large as the SPT radius of the same vertex set.

**Proof:** According to Theorem 8, the radius of every vertex in the unrestricted tangent arborescence is no more than twice that vertex's corresponding SPT radius. Because the radius of a tree is the maximum of its vertices radii, the radius of this arborescence will be no more than twice the SPT radius on the same point set. \( \Box \)

### 5.4 Point of Intersection Tangent Arborescence

For the tangent arborescence ROI, the second merge type which warrants only cursory examination is the point of intersection Steinerization. In this merge technique, the Steiner point is placed where the regions of influence intersect, which will be nearer to the root than either vertex. But, as was the case for the spanning tangent arborescence (Section 5.2), the ROI intersections tend to limit arborescence progress toward the root. Figure 5.1 shows a sample ROI intersection. Notice the point of intersection does not seem to aid progress to the root greatly.

Figure 5.8 depicts this potential problem with the arborescences resulting from the use of this merge technique. The radius of the arborescence can be an arbitrarily large factor of the SPT radius. The arborescence cost can be very large as well. Hence, this algorithm was not examined fully.
5.5 Restricted Tangent Arborescence

The usefulness of the point of intersection merge technique with the tangent arborescence ROI seems quite low after Section 5.4. However, slight modifications of the tangent arborescence ROI may exploit hidden algorithmic potential when using this merge technique.

Initial concerns when considering the tangent ROI led to the idea of an angle restriction. This angle restriction was introduced in the hope that it would provide a bound on the arborescence radius. The ROI is still defined by \( \tau_1 \cap \tau_2 \cap \tau_3 \) (as discussed in Section 5.1), but \( \tau_2 \) has the further restriction that the angle can be no more than 90° (45° on each side of the SPT edge for that vertex). The choice of 90° was somewhat arbitrary and other angle limits could be examined (see Section 7.2 for more details).

Figures 5.9(a,b,c,d) presents the changes this ROI undergoes as the sweepcircle collapses. Figure (a) depicts the case where \( \tau_2 \) is limited to 90°. Comparison of Figures 5.9(a) and 5.3(a) reveals the difference between the unrestricted and restricted versions of this algorithm. The sweepcircle eventually reaches a critical point (Figure 5.9(b)) where the more familiar tangent ROI resumes. The overall region which the ROI encompasses can be seen in Figure 5.10, and is similar to Figures 4.3 and 5.4.

The repercussions of this change to the ROI are quite significant. With two possible states for \( \tau_2 \) (one restricted to 90° and one based on tangents to the sweep-
circle), there are now four possible situations arising from the intersection of two regions of influence. These intersection events are:

- **ROI–Vertex Intersection** (Example: Figure 5.11(a))
- **Tangent–Tangent Intersection** (Example: Figure 5.11(b))
- **90°–90° Intersection** (Example: Figure 5.11(c))
- **Tangent–90° Intersection** (Example: Figure 5.11(d))
Proposition 4  The timestamp \( \overline{BR} \) for a ROI-Vertex ROI intersection is simply \( \overline{BR} \).

Proof: Vertex \( B \) has activated inside another ROI, but is still on the sweepcircle frontier. Therefore, the event occurs at time \( \overline{BR} \). □
Proposition 5  The timestamp \((\overline{CR})\) for a Tangent-Tangent ROI intersection is

\[
\overline{CR} = \cos(90^\circ - \angle BAR) \cdot \overline{AR}
\]

Proof: See Proposition 3. □

Proposition 6  The timestamp \((\overline{CR})\) for a 90°-90° intersection is

\[
\overline{CR} = \sqrt{AC^2 + AR^2} - \sqrt{2 \cdot AC \cdot AR}
\]

where

\[
\overline{AC} = \frac{\overline{AB} \cdot \sin(\angle ABR - 45^\circ)}{\sin(270^\circ - \angle BAR - \angle ABR)}
\]

Proof: Consider Figure 5.11(c).

1. Segments \( \overline{AR} \) and \( \overline{BR} \) bisect their respective regions of influence.
   Resulting from the restricted tangent ROI definition.
2. \( \angle CAR = \angle CBR = 45^\circ \)
   The restricted tangent ROI definition has an angle restricted to 90°.
   The segments mentioned in step 1 bisect each ROI.
3. \( \angle BAC = \angle BAR - \angle CAR = \angle BAR - 45^\circ \)
   Angle addition and subtraction principles.
4. \( \angle ABC = \angle ABR - \angle CBR = \angle ABR - 45^\circ \)
   Angle addition and subtraction principles.
5. \( \angle ACB = 180^\circ - \angle BAC - \angle ABC \)
   The interior angles of a triangle sum to 180°.
6. \( \angle ACB = 270^\circ - \angle BAR - \angle ABR \)
   Substitution of steps 3 and 4 into step 5.
7. \[ \overline{AC} = \frac{\overline{AB} \cdot \sin(\angle ABR - 45^\circ)}{\sin(270^\circ - \angle BAR - \angle ABR)} \]

Law of Sines.

8. \[ \cos \angle CAR = \cos 45^\circ = \frac{1}{\sqrt{2}} \]

From step 2.

9. \[ \overline{CR} = \sqrt{\overline{AC}^2 + \overline{AR}^2 - 2 \cdot \overline{AC} \cdot \overline{AR} \cdot \cos \angle CAR} \]

Law of Cosines.

10. \[ \overline{CR} = \sqrt{\overline{AC}^2 + \overline{AR}^2 - \sqrt{2} \cdot \overline{AC} \cdot \overline{AR}} \]

Substitution of step 8 into step 9.

\[ \square \]

**Proposition 7** The timestamp (\( \overline{CR} \)) for a Tangent-90° intersection where B’s ROI is restricted to 90° is

\[ \overline{CR} = \sqrt{\overline{BC}^2 + \overline{BR}^2 - \sqrt{2} \cdot \overline{BC} \cdot \overline{BR}} \]

where

\[ \overline{BC} = \min(\frac{-t_1 + \sqrt{t_1^2 - 8 \cdot t_2}}{4}, \frac{-t_1 - \sqrt{t_1^2 - 8 \cdot t_2}}{4}) \]

\[ t_1 = -2 \cdot \overline{AB} \cdot \cos(\angle ABR - 45^\circ) - \sqrt{2} \cdot \overline{BR} \]

\[ t_2 = \overline{AB}^2 + \overline{BR}^2 - \overline{AR}^2 \]

**Proof:** Consider Figure 5.11(d).

1. \( \angle CBR = 45^\circ \)

   The ROI for vertex B is limited to an angle of 90°.

   The segment \( \overline{BR} \) bisects the ROI for vertex B.

2. \( \angle CBA = \angle ABR - \angle CBR = \angle ABR - 45^\circ \)

   Angle addition principles.
3. \( \overline{AR}^2 = \overline{AC}^2 + \overline{CR}^2 \)

Pythagorean theorem.

4. \( \overline{AC}^2 = \overline{AB}^2 + \overline{BC}^2 - 2 \cdot \overline{AB} \cdot \overline{BC} \cos \angle CBA \)

Law of Cosines.

5. \( \overline{CR}^2 = \overline{BR}^2 + \overline{BC}^2 - 2 \cdot \overline{BR} \cdot \overline{BC} \cos 45^\circ \)

Law of Cosines.

6. \( \overline{CR} = \sqrt{\overline{BC}^2 + \overline{BR}^2 - 2 \cdot \overline{BC} \cdot \overline{BR}} \)

Simplification of step 5.

**Note:** Some unknown terms remain in step 6. These will now be determined.

7. \( 2\overline{BC}^2 - 2(\overline{AB} \cdot \cos \angle CBA + \overline{BR} \cdot \cos 45^\circ)\overline{BC} + \overline{AB}^2 + \overline{BR}^2 - \overline{AR}^2 = 0 \)

Substitution of steps 4 and 5 into step 3.

For convenience, let

\[
\begin{align*}
t_1 &= -2(\overline{AB} \cdot \cos \angle CBA + \overline{BR} \cdot \cos 45^\circ) \\
    &= -2 \cdot \overline{AB} \cdot \cos(\angle ABR - 45^\circ) - \sqrt{2} \cdot \overline{BR} \\
t_2 &= \overline{AB}^2 + \overline{BR}^2 - \overline{AR}^2
\end{align*}
\]

8. \( \overline{BC} = \min\left(\frac{-t_1 + \sqrt{t_1^2 - 4 \cdot t_2}}{2}, \frac{-t_1 - \sqrt{t_1^2 - 4 \cdot t_2}}{2}\right) \)

Application of the Quadratic formula on step 7.
In step 8, the derivation of $BC$ uses the minimum of two roots of a quadratic equation, but this rationale has not yet been explained. Consider Figure 5.12, which is similar to part of Figure 5.11(d). It will now be shown that the location of point $C$ within the diagram could be in one of two places. Given that the ROI for $A$ is known to be a tangent ROI, and that the point of tangency forms the apex of a right angle between vertices $A$ and $R$, vertex $C$ could be at point $C_1$ or point $C_2$. Clearly the point intersection will occur at the member of \{ $C_1, C_2$ \} which is nearest vertex $B$. Therefore, the minimum of the two solutions to $BC$ is used.

Note that the timestamp for each ROI Intersection case can be determined quickly, however each case must also be easily identified as the algorithm progresses. Given that the sweepcircle adjacency cycle has identified two adjacent regions of influence (associated with vertices $i$ and $j$), the ROI intersection case must be correctly determined to compute the proper timestamp. To do this, assume (without loss of generality) that vertex $i$ is the vertex from this pair which is closest to the root. Given the position of vertex $i$ and the root $r$, where could the remaining vertex $j$ be? Consider Figure 5.13(a), depicting the position of $i$, $r$ and various boundaries to be discussed.

First, note that because vertex $i$ was assumed to be closer to $r$ than vertex $j$, vertex $j$ cannot be inside the circle centered at $r$ with a radius $ir$.

Second, the ROI-vertex intersection case (as depicted in Figure 5.11(a)) is considered. In this case, no Steiner point is necessary as the point of intersection occurs at vertex $i$. In order for this to occur, $\angle ijr \leq 45^\circ$ otherwise vertex $i$ would never fall inside the ROI for vertex $j$. Additionally, vertex $j$ must be located where, as its ROI expands, vertex $i$ will fall inside it. If $\angle ijr \geq 90^\circ$, then vertex $i$ is guaranteed to fall inside the ROI for vertex $j$. Therefore if $\angle ijr \leq 45^\circ$ and $\angle ijr \geq 90^\circ$, then the ROI for vertex $j$ will intersect with vertex $i$. The location of $j$ is defined by the intersection of these two inequalities. The region defined by $\angle ijr \leq 45^\circ$ is the exterior of the circles labeled $B$ in Figure 5.13(b), and the region above line $A$ in the same figure is where $\angle ijr \geq 90^\circ$. Thus, if vertex $j$ is located in the shaded region of
Figure 5.13: Determining the ROI Intersection Case

Figure 5.13(c), then \( i \) and \( j \) will have a ROI-vertex intersection.

The location of \( j \) in a tangent-tangent ROI intersection is quite simple to determine. In order for two tangent regions of influence to intersect, each must be inside the others ROI. In other words, \( \angle rji \leq 45^\circ \) and \( \angle rijd \leq 45^\circ \). The first inequality once again describes the exterior of the circles labeled \( B \) in Figure 5.13(b). The second inequality translates into the interior of the angle limited by the lines labeled \( C \) in
Figure 5.13(b). The intersection of these two regions defines where \( j \) can be located, and is presented as the shaded area shown in Figure 5.13(d).

For \( i \) and \( j \) to be involved in a \( 90^\circ-90^\circ \) ROI intersection, vertex \( i \) cannot lie inside the angle defining \( j \)'s ROI and vice versa. As such, \( \angle ij r \geq 45^\circ \) and \( \angle j ir \geq 45^\circ \). This means that vertex \( j \) must be inside the circles labeled \( B \) in Figure 5.13(b). As well, vertex \( j \) must also be close enough to \( i \) so that \( j \) remains a \( 90^\circ \) angle restricted ROI. This is enforced by making sure the distance between \( j \) and segment \( \overline{ir} \) is less than \( dist(i, r) \). This region associated with this latter restriction is the area between the two segments labeled \( D \) in Figure 5.13(b). The result is the shaded region of Figure 5.13(e), representing the possible locations of \( j \) when a \( 90^\circ-90^\circ \) ROI intersection event occurs.

Finally, position of \( j \) for the tangent-\( 90^\circ \) intersection event must be determined. The remaining area constitutes the possible locations of \( j \) for this intersection event, and is displayed as the shaded region of Figure 5.13(f).

5.5.1 The Algorithm

The following is the pseudo-code for the restricted tangent arborescence algorithm.

<table>
<thead>
<tr>
<th>Given a set, ( S ), of ( n ) points in the Euclidean plane, including one special point, ( r ), also known as the root.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>[Preprocessing]</strong></td>
</tr>
<tr>
<td>Sweepcircle-Adjacency.Initialize.</td>
</tr>
<tr>
<td>Event-Queue.Initialize</td>
</tr>
<tr>
<td>For each ( i ), where ( i \in S - { r } )</td>
</tr>
<tr>
<td>Set ( i.\text{State} = \text{Inactive} ).</td>
</tr>
<tr>
<td>Event-Queue.Insert(( Point \ Activation,{ i },dist(i, r) ))</td>
</tr>
<tr>
<td><strong>[Main Loop]</strong></td>
</tr>
<tr>
<td>While not Event-Queue.Empty do</td>
</tr>
<tr>
<td>Case Event-Queue.Remove of:</td>
</tr>
</tbody>
</table>

82
**Point Activation** Event for vertex \( i \)

Set \( i.\text{State} = \text{Active} \)

\[
\text{Sweepcircle-Adjacency.Insert}(i)
\]

For each \( j \in \text{Sweepcircle-Adjacency.Neighbors}(i) \)

\[
\text{Timestamp} = \text{Compute-Timestamp}(i, j, r)
\]

\[
\text{Event-Queue.Insert}(\text{ROI Intersection}, \{i, j\}, \text{Timestamp})
\]

\[
\text{Event-Queue.Insert}(\text{ROI Intersection}, \{i, r\}, 0.0)
\]

**ROI Intersection** Event between vertices \( i \) and \( j \)

If \( i.\text{State} \neq \text{Deactivated} \) and \( j.\text{State} \neq \text{Deactivated} \)

[Ignore intersection events involving deactivated points.]

then

If \( ((\text{Timestamp} = \text{dist}(i, r)) \) or \( (\text{Timestamp} = \text{dist}(j, r)) \))

then

[New point activation inside existing ROI]

if \( \text{dist}(i, r) > \text{dist}(j, r) \)

then

\[
\text{near} = j
\]

\[
\text{far} = i
\]

else

\[
\text{near} = i
\]

\[
\text{far} = j
\]

Set \( \text{far}.\text{State} = \text{Deactivated} \).

Add edge \((\text{far}, \text{near})\) to arborescence.

\( m = \text{Sweepcircle-Adjacency.Neighbors}(\text{far}) \) such that

\[
\text{near} \notin m
\]

\[
\text{Timestamp} = \text{Compute-Timestamp}(\text{near}, m, r)
\]

\[
\text{Event-Queue.Insert}(\text{ROI Intersection}, \{\text{near}, m\}, \text{Timestamp})
\]

\[
\text{Sweepcircle-Adjacency.Remove}(\text{far})
\]
else

Add Steiner point \( k \) to the vertex set.

Set \( i.\text{State} = \text{Deactivated} \).

Set \( j.\text{State} = \text{Deactivated} \).

Add edges \((i, k)\) and \((j, k)\) to arborescence.

Event-Queue.Insert(\( \text{Point Activation,} \{k\}, \text{dist}(k, r) \))

\[ m_1 = \text{Sweepcircle-Adjacency.Neighbors}(i) \text{ such that} \]
\[ j \notin m_1 \]

\[ m_2 = \text{Sweepcircle-Adjacency.Neighbors}(j) \text{ such that} \]
\[ i \notin m_2 \]

\( \text{Timestamp} = \text{Compute-Timestamp}(m_1, m_2, r) \)

Event-Queue.Insert(\( \text{ROI Intersection,} m_1 \cup m_2, \text{Timestamp} \))

Sweepcircle-Adjacency.Remove(\( i \))

Sweepcircle-Adjacency.Remove(\( j \))

[Postprocessing]

Output arborescence.

The routine \( \text{Compute-Timestamp}(i, j, r) \) determines the timestamp for the intersection of the regions of influence for vertices \( i \) and \( j \) with respect to the root, \( r \).

\begin{align*}
\text{Compute-Timestamp}(i, j, r): \\
\text{If} \ ((\angle ijr \geq 90^\circ \text{ and } \angle rij \leq 45^\circ)) \text{ or} \\
\quad ((\angle jir \geq 90^\circ \text{ and } \angle rji \leq 45^\circ)) \\
\text{then} \\
\quad \text{[No Steiner point necessary–ROI and Vertex intersection]} \\
\quad \text{return} \ \min(\text{dist}(i, r), \text{dist}(j, r)) \\
\text{else}
\end{align*}
If \((\angle ijr \leq 45^\circ)\) and \((\angle jir \leq 45^\circ)\)
then

\([\text{Tangent–Tangent intersection}]\)

\[
\text{return } \cos(90^\circ - \angle rij) \cdot \text{dist}(i, r)
\]

else

\([\text{Find }\text{ temp, the distance from }j\text{ to segment }ir]\)

\[
\text{If } (ir < jr)
\]
then

\[
\text{If } (\angle jir \leq 90^\circ)
\]
then

\[
\text{temp} = ij \cdot \sin \angle jir
\]
else

\[
\text{temp} = ij \cdot \sin(180^\circ - \angle jir)
\]
else

\[
\text{If } (\angle ijr \leq 90^\circ)
\]
then

\[
\text{temp} = ij \cdot \sin \angle ijr
\]
else

\[
\text{temp} = ij \cdot \sin(180^\circ - \angle ijr)
\]
If \((\angle ijr \geq 45^\circ)\) and \((\angle jir \geq 45^\circ)\)
and \((\text{temp} < \text{dist}(i, r))\)
then

\([90^\circ-90^\circ \text{ intersection}]\)

\[
\overline{AC} = \frac{\text{dist}(i,j) \cdot \sin \angle rji}{\sin(270^\circ - \angle rij - \angle rji)}
\]

\[
\text{return } \sqrt{\overline{AC}^2 + \text{dist}(i, r)^2} - \sqrt{2} \cdot \overline{AC} \cdot \text{dist}(i, r)
\]
else

\[
\text{If } (\text{dist}(i, r) > \text{dist}(j, r))
\]
then

[Tangent–90° intersection]

\[
t_1 = -2 \cdot \text{dist}(i, j) \cos(\angle ijr - 45°) - \text{dist}(j, r)\sqrt{2}
\]

\[
t_2 = \text{dist}(i, j)^2 + \text{dist}(j, r)^2 - \text{dist}(i, r)^2
\]

\[
BC = \min(-t_1 \pm \sqrt{t_1^2 - 8t_2})
\]

return \[\sqrt{BC^2 + \text{dist}(j, r)^2 - \sqrt{2} \cdot BC \cdot \text{dist}(j, r)}\]

else

[90°–Tangent intersection]

\[
t_1 = -2 \cdot \text{dist}(j, i) \cos(\angle jir - 45°) - \text{dist}(i, r)\sqrt{2}
\]

\[
t_2 = \text{dist}(j, i)^2 + \text{dist}(i, r)^2 - \text{dist}(j, r)^2
\]

\[
BC = \min(-t_1 \pm \sqrt{t_1^2 - 8t_2})
\]

return \[\sqrt{BC^2 + \text{dist}(i, r)^2 - \sqrt{2} \cdot BC \cdot \text{dist}(i, r)}\]

5.5.2 Algorithm Analysis

5.5.2.1 Time Complexity Analysis

As was the case for the other algorithms presented here, the restricted tangent arborescence algorithm is divided into three sections. The preprocessing section of the algorithm performs some initialization for each vertex, including an insertion of its point activation event. Each point activation event insertion requires \(O(\log n)\) time to complete, so the preprocessing portion of this algorithm requires \(O(n \log n)\) total time. The postprocessing section is identical to all the other algorithms, requiring \(O(n)\) time.

The processing of each point activation event in the main loop of the algorithm generates three ROI intersection events, an insertion into the sweepcircle adjacency cycle, and a query to the sweepcircle adjacency cycle. Each of these require \(O(\log n)\) time, so the \(O(n)\) point activation events will require \(O(n \log n)\) total time.

When processing ROI intersection events there are two cases to be considered: one involves the addition of a Steiner point to the vertex set, the other does not.
If a Steiner point is added, there are two event insertions made, two queries to the sweepcircle adjacency cycle, and two updates to the sweepcircle adjacency cycle. Each of these requires $O(\log n)$ time. If the merge is performed using only existing vertices, one event insertion is done, along with one sweepcircle query and one sweepcircle adjacency cycle update. Once again, each requires only $O(\log n)$ time. Since there can be only $O(n)$ ROI intersections that actually require processing, the main loop requires $O(n \log n)$ time.

The Compute-Timestamp() routine involves strictly constant time computations.

The restricted tangent arborescence algorithm therefore requires $O(n \log n)$ total time to complete.

5.5.2.2 Space Requirements Analysis

The algorithm requires at least $O(n)$ space because it needs to store all relevant vertex information.

The size of the event queue will depend on the number of point activation events and the number of ROI intersection events required. Each vertex, including Steiner points, will require a point activation event, for a total of $O(n)$ of those. When each vertex activates or deactivates there are a constant number of ROI intersection events added to the event queue. Each of the $O(n)$ vertices must both activate and later deactivate, so there will be $O(n)$ ROI intersection events. Each event requires a constant amount of space, for a total of $O(n)$ space.

The sweepcircle adjacency graph contains information for every vertex. There are $O(n)$ vertices, and each requires a constant amount of space. So the space requirements of the restricted tangent arborescence algorithm is $O(n)$.

5.5.2.3 Algorithm Properties

A bound for the $cost_{ratio}$ metric for the restricted tangent arborescence algorithm is currently an open problem.
**Theorem 10** The radius of any vertex \( i \) in a restricted tangent arborescence is no more than \((1 + \sqrt{2}) \cdot i \bar{r}\), where \( r \) is the root of the arborescence.

**Proof:** By induction on the number of intermediate vertices on the path between \( v_i \) and \( r \).

![Diagram](image)

**Figure 5.14:Restricted Tangent ROI Shape**

**Base Case:** Given one intermediate point \( j \) on the path between \( i \) and \( r \). The path will contain 2 edges, call them \( a \) and \( b \). Where will \( j \) be to maximize \((a + b)\)?

Consider Figure 5.14. The shaded region is the union of all regions of influence. The vertex \( j \) may be located anywhere within the boundaries of this region. The region is divided into two portions, based on the state of \( \tau_2 \). When the ROI angle is limited to 90°, most of the top portion is generated. When the ROI angle is based on the point of tangency to the sweepcircle, the remainder is generated. The location of \( j \) will be somewhere within these two regions. The worst possible location for \( j \) will be along the frontier of this region, since if \( j \) is located interior to this region, the path \( \bar{ij} + j\bar{r} \) can be made larger by simply moving \( j \) toward the nearest point on the frontier.

Assume \( j \) is on the circle whose diameter is segment \( i\bar{r} \). Clearly, \((a + b)\) will be worse when \( j \) located on this circle than if it were in a subset of this region (i.e., if it were inside the shaded region). Where will \( j \) be located on this circle to maximize \((a + b)\)? Assuming that \( i\bar{r} = 1 \), it is known that \( a = \sqrt{1 - b^2} \). To find the critical point for \((a + b)\), \( \frac{da}{db}(\sqrt{1 - b^2} + b) \) must be found, which yields \((1 - \frac{b}{\sqrt{1-b^2}})\).
Setting the derivative to zero and solving for \( b, b = \frac{1}{\sqrt{2}} \) is obtained. Therefore, \((a + b)\) reaches a critical point when \( a = b = \frac{1}{\sqrt{2}} \). By taking the second derivative of \((a + b)\) and setting the result to zero, it can be confirmed that this is indeed a maximum point. Therefore, \((a + b) = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} = \sqrt{2} \leq 1 + \sqrt{2}, \) and the base case holds.

**Inductive Hypothesis:** Assume with \( n \) intermediate vertices on the path between \( i \) and \( r \), that the sum of the edge costs on that path will be no more than \((1 + \sqrt{2}) \cdot \ell_{ir}\).

**Inductive Step:** Show this property holds with \( n + 1 \) intermediate vertices between \( i \) and \( r \).

Let the first intermediate vertex be \( j \). From the discussion above under the base case, this vertex will be on the frontier of the shaded region in Figure 5.14. Because this region is divided into two distinct portions, each will be examined independently.

If \( a = \overline{ij} \) and \( b = \overline{jr} \), then by the inductive hypothesis, it must be shown that \( a + (1 + \sqrt{2})b \leq 1 + \sqrt{2} \).

First, assume \( j \) is on the portion of the ROI limited by the 90° angle. Therefore, by the Law of Cosines, \( b = \sqrt{a^2 + 1 - 2 \cdot a \cdot \cos 45°} = \sqrt{a^2 + 1 - \sqrt{2}a} \). This new value for \( b \) is substituted into \((a + (1 + \sqrt{2})b)\) and its derivative is found: \( \frac{d}{da}(a + (1 + \sqrt{2})(\sqrt{a^2 + 1 - \sqrt{2}a})) = 1 + \frac{(1 + \sqrt{2})(2a - \sqrt{2})}{2\sqrt{1 - \sqrt{2}a}a} \). Setting this result to zero and solving for \( a \), the following is obtained: \( a = \frac{\sqrt{2} - \sqrt{2 - 1}}{2} \approx 0.385 \). Using this value for \( a \), \( a + (1 + \sqrt{2})b \approx 1.842 \). It must now be determined if this is a maximum or a minimum point. When \( a = 0 \), then \( b = 1 \) and \((a + (1 + \sqrt{2})b) = 1 + \sqrt{2} \). As well, when \( a = \frac{1}{\sqrt{2}}, b = \frac{1}{\sqrt{2}} \) as well, and \((a + (1 + \sqrt{2})b) = 1 + \sqrt{2} \). Therefore, \( a \approx 0.385 \) is a minimum, not a maximum. This implies that if \( j \) is located in the 90° restricted region, then \((a + (1 + \sqrt{2})b)\) is a maximum when \( a = \frac{1}{\sqrt{2}} \).

Alternatively, it can be assumed that \( j \) is located on the lower portion of the ROI. By the Pythagorean theorem, \( a = \sqrt{1 - b^2} \). This new value for \( a \) is substituted into \((a + (1 + \sqrt{2})b)\) and its derivative is found: \( \frac{d}{db}(\sqrt{1 - b^2} + (1 + \sqrt{2})b) = 1 + \sqrt{2} - \frac{b}{\sqrt{1 - b^2}} \). Setting this result to zero and solving for \( b \), the following is obtained: \( b = \frac{\sqrt{2 + \sqrt{2}}}{2} \).
It can be verified that \( b = \frac{\sqrt{2} + \sqrt{2}}{2} \) is indeed a maximum point. However, \( b \) must have a value such that \( 0 \leq b \leq \frac{1}{\sqrt{2}} \). Because the value for \( b \) is outside the defined range when maximized, \( b \) must be reduced into its acceptable range. With \( b = \frac{1}{\sqrt{2}} \), a corresponding value for \( a \) can be found: \( a = \frac{1}{\sqrt{2}} \). By substituting these new found values into the equation \( a + (1 + \sqrt{2})b \leq 1 + \sqrt{2} \), it can be noted that the inequality holds.

Therefore, the cost of a path to the root \( r \) from vertex \( i \) in the restricted tangent arborescence is no more than \( (1 + \sqrt{2}) \cdot \overline{ir} \). □

**Theorem 11** The radius of the arborescence generated by the restricted tangent arborescence algorithm will have a radius which is no more than \( (1 + \sqrt{2}) \) times as large as the SPT radius of the same vertex set.

**Proof:** According to Theorem 10, the radius of every vertex in a restricted tangent arborescence is no more than \( (1 + \sqrt{2}) \) times that vertex’s corresponding SPT radius. Because the radius of a tree is the maximum of its vertices radii, the radius of this arborescence will be no more than \( (1 + \sqrt{2}) \) times the SPT radius on the same point set. □

### 5.6 Summary

Two arborescence algorithms are presented in this chapter. Both are based to some extent upon a visibility property. The regions of influence for two vertices intersect when the vertices become visible to each other over the sweepcircle frontier. The first algorithm, the unrestricted tangent arborescence algorithm, uses a classic Steinerization merge technique to perform component merges. The second algorithm, the restricted tangent arborescence algorithm uses a slightly modified ROI and the point of intersection Steinerization technique. Each of these algorithms has a computational complexity of \( O(n \log n) \) time, and require \( O(n) \) space.
Chapter 6

Empirical Study

For some algorithms, provable bounds on quality may not always be possible, or provable bounds may not reflect typical quality. However, an investigation of the experimental quality of such algorithms can be undertaken to get a sense of their quality. Even in cases where quality bounds can be shown to exist, experimentation may reveal very good quality solutions are generated on average, even though worst case performance may be quite different.

This chapter contains an examination of the empirical quality of the arborescences generated by the algorithms introduced in the previous chapters.

6.1 Implementation

The implementation of each of the four algorithms was done in the C programming language under UNIX-derivative operating systems.

The implementation accepts point set input from a specified input file, or uses a random seed to generate the vertex set. The coordinates for random data sets were generated using the C random() function. Without loss of generality, the point set was assumed to be wholly contained within the unit square.

The event queue for the algorithms was implemented as a heap. The sweepcircle adjacency cycle was implemented as a binary tree and the vertex and arborescence information was maintained in arrays of records. A graphical display of the vertex set and the trees generated has been implemented under the X-Window System.
The implementation can produce six different tree configurations for a given point set:

- Circle Spanning Arborescence Algorithm (Example: Figure 6.1)
- Circle Steiner Arborescence Algorithm (Example: Figure 6.2)
- Unrestricted Tangent Arborescence Algorithm (Example: Figure 6.3)
- Restricted Tangent Arborescence Algorithm (Example: Figure 6.4)
- Minimum Spanning Tree (Example: Figure 6.5)
- Shortest Path Tree (Example: Figure 6.6)

Figure 6.1: Example Circle Spanning Arborescence
The vertex set supported has an upper limit set to 32768 vertices, which could be modified at compile time. The randomly generated vertex set can have one of three distributions:

- Uniform (Example: Figure 6.7)
- Normal (Example: Figure 6.8)
- Cluster (Example: Figure 6.9)

For the uniform distribution, $x$ and $y$ coordinates are each generated uniformly within the $[0, 1]$ interval (resulting in a data set contained within the unit square). The normal distribution generates $x$ and $y$ coordinates, each normally distributed as $N(0.5, 0.2)$. To generate the normal distribution, the Polar Method of Marsaglia and Bray [16] is used. The cluster distribution utilizes a hybrid of the uniform and the normal distributions. The point set consists of a series of independent
clusters. Each cluster has a uniformly distributed cluster center and is assigned a random proportion of the total vertex set. The vertices assigned to each cluster are distributed normally around their particular cluster center. Figures 6.7, 6.8 and 6.9 show an example of each distribution on five hundred vertices.

6.1.1 Metrics

As discussed in Section 3.8, a number of metrics can be examined to gauge arborescence quality. These metrics include:

- \(\text{cost}_{\text{ratio}}\) – Ratio between arborescence cost and MST cost.
- \(\text{radius}_{\text{ratio}}\) – Ratio between arborescence radius and SPT radius.
- \(\text{radius}_{\text{max}}_{\text{ratio}}\) – Maximum observed ratio between a vertex's radius in arborescence and that vertex's radius in the SPT.
In turn, given multiple observations of each of these metrics, there are a number of interesting statistics which can be obtained from each of these metrics:

- **Minimum** – the smallest measurement of a particular metric observed.
- **Maximum** – the largest measurement of a particular metric observed.
- **Mean** – the average measurement of the particular metric.
- **Coefficient of Variation** – a measure of observation dispersion; how well the mean of a metric characterizes a typical observation.\(^1\)

---

\(^1\)The **coefficient of variation** (C.O.V.) is determined by \(\frac{s}{\bar{x}}\) where \(s\) is the sample standard deviation and \(\bar{x}\) is the sample mean of the observations. A C.O.V. of 5 is large, and a C.O.V. of 0.2 is small[14]. A very low C.O.V. implies that the sample mean provides almost the same information as the complete set of measurements[14].
6.2 Experimental Design

To gather the metrics discussed above, a series of experiments were performed. A full factorial experimental design was used. There are three factors to be examined: number of vertices (problem size), algorithm and vertex distribution. The size factor consists of ten levels: 5, 10, 25, 50, 100, 500, 1000, 2500, 5000 and 10000 vertices. There are four algorithms and three vertex distribution methods. This yields a total of 120 trials to be conducted.

The sample size for each trial was limited by timing considerations. For trials using size levels of 5, 10 and 25, ten thousand random vertex sets were generated for each trial. Sizes 50, 100 and 500 were tested on one thousand random point sets. Experiments for sizes 1000 and 2500 consisted of one hundred samples, and sizes 5000 and 10000 used 50 random data sets. Due to the computational complexity of the circle arborescence algorithms (as per Sections 4.2.2.1 and 4.3.2.1), which translated
Figure 6.6: Example Shortest Path Tree

Figure 6.7: Uniform Vertex Distribution
int into lengthy computation times during experimentation, these two algorithms were not examined with point sets of size 10000.

Because the sample sizes for the various trials vary by so much (50 to 10,000 samples per trial) this alone may have had a significant impact upon experimental
results. This potential problem was tested for by reducing the sample size for each trial to 50. The graphs and experimental results obtained were virtually identical. Some extreme (minimum and maximum) values were slightly different, but overall the results were the same.

6.3 Measurements

This section will examine the results of the experiments described above. Graphs depicting the various metrics will be the primary means of communicating the experimentally observed characteristics of each algorithm.

6.3.1 Arborescence Cost

The first task is to examine the effect of vertex distribution on the cost of the resulting arborescence.

Figure 6.10 depicts the spread of observed cost ratio measures for the various distributions in the circle spanning arborescence algorithm. Each distribution appears fairly similar. Each starts with an almost identical average and minimum value, as well as a relatively expensive cost ratio. However, the maximum quickly begins to converge on the average and minimum observed value. Overall it appears that each distribution seems to follow similar trends as far as the maximum, average and minimum cost ratio metric is concerned.

Figure 6.11(a) presents the effect the choice of distribution has on the average cost ratio metric for the circle spanning algorithm. On very small vertex sets, the choice of distribution is less significant, due to the relative similarity between each on very few points. However, on larger vertex sets, the normal distribution has a cost closer to the MST cost on average. This is somewhat logical, since the vertices on the periphery of the single large cluster will have MST edges which, in general, move toward the center. Since there are relatively few vertices in the periphery, the nearest vertex will probably be in the direction of the root. So both the MST
Figure 6.10: Circle Spanning cost\textsubscript{ratio} Over Various Distributions

Figure 6.11: Comparing Distributions for Circle Spanning Algorithm
and the arborescence on a normally distributed vertex set will tend to have edges moving towards the root (which is located at the mean of the normal distribution).

By the same token, this argument explains why the cluster distribution’s $cost_{ratio}$ is relatively poor. Vertices on the periphery of a normal distribution will tend to move towards the mean of the normal curve. As well, the cluster distribution is a set of uniformly distributed normal clusters. Therefore, the MST on a cluster distribution will have vertices which tend to add edges towards the center of their respective cluster, whereas the peelable property of these arborescences have vertices which tend to add edges towards the root. If a cluster center and the root lie in opposing directions from a vertex, the MST and arborescence cost will begin to diverge.

Figure 6.11(a) would lead to the belief the circle spanning arborescence produces trees which are reasonably close to the MST cost, regardless of the distribution. Figure 6.11(b) shows that these averages have a low C.O.V. Therefore, the averages depicted in Figure 6.11(a) are indicative of the sample as a whole, and therefore, are probably representative of the population. Similar analysis and conclusions can be drawn from the data with respect to the other algorithms: point set distributions do not significantly affect the performance of the algorithms with respect to the $cost_{ratio}$ metric.

The graphs in Figure 6.12 present the range of $cost_{ratio}$ values observed for each algorithm during experimentation when a uniform point distribution was used. Three of the algorithms (circle spanning, circle steiner and restricted tangent) have very similar plots. A fairly wide range of $cost_{ratio}$ values were observed for small point sets, which quickly converges when large vertex sets are used. The odd algorithm out, the unrestricted tangent algorithm, seems to perform well initially, but quickly becomes quite poor. On the larger point sets (5000 and 10000 vertices), the arborescences produced are over three times the MST cost.

Figure 6.13(a) summarizes the $cost_{ratio}$ metric for the various algorithms under the uniform vertex distribution. The most striking portion of this graph is the poor $cost_{ratio}$ performance of the unrestricted tangent arborescence relative to the
Figure 6.12: Algorithm Ranges for cost\textsubscript{ratio} Metric

Figure 6.13: Arborescence Quality by Algorithm on a Uniform Distribution
other algorithms. This aside, the other algorithms all seem to have very respectable \( cost_{ratio} \) performance. Performance under this metric seems to become quite stable as the point set grows. The flatness of the graph on large vertex sets is compounded when the logarithmic scale for the problem size (x-axis) is noted. Both the circle-based ROI algorithms perform superbly under this metric, averaging about 15% higher cost than the MST. The restricted tangent algorithm performs very well also, averaging only about 25% higher cost than the MST on large point sets.

Figure 6.13(b) plots the coefficient of variation information of the \( cost_{ratio} \) for each algorithm. Once again, the low nature of the C.O.V. indicates that each mean as displayed in Figure 6.13(a) is representative of the experimental observations as a whole.

Unlike the other two metrics, proofs regarding worst-case arborescence quality with respect to \( cost_{ratio} \) are open. The growth of the \( cost_{ratio} \) metric for the unrestricted tangent arborescence does not show any signs of slowing. The other three algorithms all had their observed worst-case \( cost_{ratio} \) performance on fairly small point sets. The restricted tangent arborescence algorithm had a maximum observed \( cost_{ratio} \) of 1.602. The circle spanning arborescence algorithm had a maximum observed \( cost_{ratio} \) of 1.434. And finally, the circle Steiner arborescence algorithm had an excellent maximum observed \( cost_{ratio} \), only 1.283. It appears that three of the algorithms perform very well in practice with respect to the \( cost_{ratio} \) metric.

### 6.3.2 Arborescence Radius

This study of arborescence radius quality will begin with an examination of the effect of distribution on this metric.

Figures 6.14(a), (b) and (c) show the range of \( radius_{ratio} \) values observed during experimentation with the circle spanning arborescence algorithm. Except for the occasional peculiarity in a maximum or minimum value, each of the graphs are extremely similar, tending to converge toward a similar average \( radius_{ratio} \).
Figure 6.14: Circle Spanning $radius_{ratio}$ Over Various Distributions

(a) Uniform Distribution  (b) Normal Distribution  

(c) Cluster Distribution

Figure 6.15: Comparing Distributions for Circle Spanning Algorithm

(a) Average $radius_{ratio}$  

(b) C.O.V. for $radius_{ratio}$
When examining the average radius of each distribution (see Figure 6.15(a)), notice that the uniform and cluster distributions tend to have virtually identical mean $radius_{ratio}$ values. The normal distribution has a mean which is consistently less, but never much more than ten percent lower. The C.O.V. graph presented in Figure 6.15(b) shows that once again, the observed averages from Figure 6.15(a) are fairly representative of the sample as a whole. Therefore, aside from a slight shift up or down, the uniform vertex distribution yields arborescences with typical $radius_{ratio}$ values. Similar analysis can be performed for each of the other algorithms, revealing that vertex distribution has little effect with the other algorithms as well.

![Graphs](image)

Figure 6.16: Algorithm Ranges for $radius_{ratio}$ Metric

The graphs in Figure 6.16 present the range of $radius_{ratio}$ values observed for each algorithm during experimentation on uniformly distributed vertices. As was
the case for cost\(\text{ratio}\), all plots except the unrestricted tangent algorithm have a similar structure: a relatively wide range of radius\(\text{ratio}\) values for small point sets, which quickly converges when large vertex sets are used. The circle spanning algorithm suffers the least convergence. The circle steiner algorithm has the smallest variability in radius\(\text{ratio}\) range. The restricted tangent algorithm has an observed range which becomes very close to the observed mean very quickly. However, the unrestricted tangent algorithm seems to perform quite poorly. As the size of the point sets increase, the range of quality of the arborescences produced is converging on the observed mean, but the observed mean itself seems to be approaching the 2-approximation of the SPT bound quite quickly. The other algorithms have observed means which quickly level out, and even decline in some cases.

![Graphs showing average radius\(\text{ratio}\) and C.O.V. for radius\(\text{ratio}\)](image)

(a) Average radius\(\text{ratio}\)  
(b) C.O.V. for radius\(\text{ratio}\)

Figure 6.17: Arborescence Quality By Algorithm On A Uniform Distribution

The quality of each algorithm under the radius\(\text{ratio}\) metric is the subject of Figure 6.17. In part (a), the observed mean radius\(\text{ratio}\) for each algorithm is shown. As was the case for the cost\(\text{ratio}\) analysis, the most striking portion of the figure is the relatively poor performance of the unrestricted tangent arborescence algorithm in comparison to the others. That algorithm starts off well enough, but it becomes the poorest of those studied in very short order. Of the remaining three algorithms, the circle Steiner algorithm consistently outperforms the others. Its plot levels off very quickly, obtaining a radius\(\text{ratio}\) sample mean of approximately 1.266 for 5000
vertices. After some initial growth, the plots for the circle spanning and restricted
tangent algorithms both seem to be converging toward the plot of the circle Steiner
algorithm. The relative flatness of these plots, especially when noting the a log-
arithmic scale used for the x-axis, is a testament to the relative stability of their
observed $radius_{ratio}$ means.

The C.O.V. of the observed $radius_{ratio}$ means is plotted in Figure 6.17(b). Once
again, the very low nature of these measurements signal that the mean measurements
in (a) are indicative of the behavior of the overall sample. Given that the samples
are probably fairly representative of the population (i.e., of all radii measurements),
the quality of three of the algorithms with respect to the $radius_{ratio}$ metric is quite
good.

### 6.3.3 Arborescence Maximum Vertex Radius

This section will study the observations of the $radius_{max_ratio}$ metric discussed in
Section 3.8. Recall from that section that this metric differs from $radius_{ratio}$ in that
the radius ratio for every vertex is considered.

Again, this study of arborescence maximum vertex radius quality will begin with
an examination of the effect of distribution on arborescence maximum vertex radius
for the circle spanning arborescence algorithm.

Figures 6.18(a), (b) and (c) show the range of $radius_{max_ratio}$ values observed
during experimentation. They are all extremely similar to each other. Each have
an observed upper bound of twice the SPT distance, as enforced by the CurDist
 technique (Section 3.6.2.1).

Comparing the average observed value for each algorithm results in Figure 6.19(a).
Notice that the three plots are all virtually identical. As well, the coefficient of vari-
ation plots in Figure 6.19(b) are virtually identical as well. An examination of
the other algorithms with respect to the $radius_{max_ratio}$ metric has been performed.
Again, it appears that the distribution does not significantly affect the current met-
ic.
(a) Uniform Distribution  (b) Normal Distribution

(c) Cluster Distribution

Figure 6.18: Circle Spanning \( \text{radius}_{\text{max ratio}} \) Over Various Distributions

(a) Average \( \text{radius}_{\text{max ratio}} \)  (b) C.O.V. for \( \text{radius}_{\text{max ratio}} \)

Figure 6.19: Comparing Distributions for the Circle Spanning Algorithm
The graphs in Figure 6.20 present the range of $radius_{\text{max ratio}}$ values observed for each algorithm during experimentation. This time, the range for each algorithm seems to have a different shape. The minimum plot seems to be a similar shape for all but the unrestricted tangent algorithm. The primary difference lies in the maximum plot.

As discussed earlier, the circle spanning arborescence algorithm seen in (a) appears to need the CurDist technique to bound the size of $radius_{\text{max ratio}}$. The maximum plot for the circle Steiner arborescence algorithm in (b) seems to perform quite well, although the range does not seem to be converging toward the observed average very quickly. The restricted tangent arborescence algorithm seen in (d) was the only algorithm which had a proven $radius_{\text{ratio}}$ bound that was greater than 2.
it was \(1 + \sqrt{2} \approx 2.414\), and it pushed beyond that 2 bound in almost every size trial. The unrestricted tangent arborescence algorithm performed better on average than the restricted tangent algorithm on all but the larger point sets. The maximum \(\text{radius}_{\text{max ratio}}\) observed for the restricted tangent arborescence is significantly larger than that of the unrestricted tangent algorithm. Although the range of the observations for the unrestricted tangent algorithm were quite tight, the quality with respect to this metric quickly began to approach 2.

Figure 6.21: Arborescence Quality By Algorithm On A Uniform Distribution

Figure 6.21(a) compares the \(\text{radius}_{\text{max ratio}}\) for all the algorithms. Once again, the unrestricted tangent algorithm performance is rather striking. It begins quite well, but quickly loses ground to all the other algorithms. Each of the other algorithms have an observed average which, after a brief increase, levels off into a quite flat curve. Again, the change in \(\text{radius}_{\text{max ratio}}\) compared to the change in problem size is overstated, due to the logarithmic scale on the x-axis. In other words, these curves are a lot flatter than they appear. The circle Steiner algorithm performs remarkably well, outperforming all algorithms by a fairly wide margin on the larger point sets.

The C.O.V. plot in Figure 6.21(b) again leads to the conclusion that the average values for \(\text{radius}_{\text{max ratio}}\) seen in (a) are representative of the entire sample. Therefore, these averages are probably representative of typical algorithm performance.
6.4 Arborescence Algorithm Quality

The low coefficients of variation for each of the measurements performed during experimentation implies that the mean values for each metric are representative of the sample as a whole. Therefore, discussion herein will be with respect to the sample mean of each metric.

Consider Figure 6.22. Overall, the circle spanning, circle Steiner and restricted tangent algorithms performed quite well. The circle Steiner seems to be the best overall algorithm. It placed second (almost tied for first) according to the cost\textsubscript{ratio} metric, and outperformed all other algorithms in both radius metrics (by a wide margin on the radius\textsubscript{maxratio} metric).

![Figure 6.22: Comparing Algorithms: cost\textsubscript{ratio} Versus radius\textsubscript{ratio}](image)

The circle spanning algorithm placed first when considering the cost\textsubscript{ratio} metric, and performed well on the radius\textsubscript{ratio} metric. When measuring the radius\textsubscript{maxratio} metric, however, it did not fare as well, as many trials apparently came close to the SPT approximation bound.

The restricted tangent algorithm also performed very well overall, obtaining reasonably good cost\textsubscript{ratio} scores. Under the radius\textsubscript{ratio} metric, it was highest for small point sets, but quickly made up ground surpassing the circle spanning algorithm by the 5000 vertex problem size. On the radius\textsubscript{maxratio} metric, it performed respectably, considering its poorer theoretical radius bound.
The unrestricted tangent algorithm performed abysmally on most of the metrics. Its growth under the cost\textsubscript{ratio} metric showed no signs of slowing. Under each of the radius metrics, the algorithm started quite well, but quickly fell off the pace and appears to be asymptotically approaching the theoretical 2 SPT approximation bound.

In practice the difference in the relative speed of the algorithms was quite noticeable, especially on large point sets. (Note: times reported here are strictly for relative comparison.) Figure 6.23 presents the results of the timing comparison performed. Notice that both of the axes have logarithmic scales. Clearly the algorithms are divided into two distinct pairs: the circle arborescence algorithms and the tangent arborescence algorithms. For very small point sets, the resolution of the clock used to time the CPU usage was not fine enough to pick up variations. Therefore, most plots begin to appear on point sets of size 25.

Impressively, the tangent arborescence algorithms required between 10 and 14 seconds even when the problem set had 10,000 vertices. On the other hand, the circle arborescence algorithms had timings which were almost two orders of magnitude larger than the tangent arborescence algorithms on the same sized problem. It must be noted however, that this implementation of the circle arborescence algorithms did not use the classic $O(n \log n)$ Voronoi Diagram algorithm, but rather an $O(n^2)$ algorithm.

![Figure 6.23: Timing Algorithm Completion](image-url)
6.5 Summary

The circle spanning, circle Steiner and restricted tangent algorithms performed quite well under the studied metrics. The circle Steiner algorithm had the best overall performance. The circle spanning algorithm had very respectable scores, and is a good choice in applications where the use of Steiner points is prohibited.

The restricted tangent algorithm also performed quite well. It was not the best according to any of the metrics, but seemed to perform quite well for all with no major weaknesses. In fact, the major weakness of the circle arborescence algorithms is the restricted tangent algorithm’s strength: computational complexity and speed. Although, the circle arborescence algorithms performed well, their $O(n^2 \log n)$ complexity in theory make them much more expensive for large problems. The $O(n \log n)$ time restricted tangent algorithm should make it a better choice on large point sets. Personal observation and some preliminary experimentation seemed to confirm these expectations, with the circle arborescence algorithms requiring much more computation time on large point sets.

The unrestricted tangent algorithm consistently performed poorly on all the metrics. Quality seemed to become poorer as the problem size grew. The algorithm is much faster than the circle arborescence algorithms on large point sets, but the arborescences produced are of very low quality.
Chapter 7

Conclusion

This thesis has presented a study of arborescences, in particular algorithms and algorithmic techniques for generating arborescences. A summary of the work done, contributions of this thesis and suggested directions for future research round out this final chapter.

7.1 Summary and Contributions

This thesis consists of three distinct parts. The first develops a modular framework for developing arborescence algorithms. The second portion utilizes this framework to produce two arborescence algorithm classes, with two variants within each, for a total of four algorithms. The final portion analyzes the quality of these four algorithms, including both theoretical and experimental quality with respect to cost and radii metrics.

The arborescence algorithm framework developed in the first portion of this thesis is a flexible and powerful tool for producing arborescence algorithms. Central to the framework and its efficiency is the sweepcircle.

The sweepcircle is an algorithmic tool utilized in the algorithm framework. It was a crucial tool used in all algorithms generated by the framework, since the natural ordering of the vertices provided by the sweepcircle allows efficient arborescence generation. Where sweepline techniques have proven useful to many problems, the sweepcircle has received little attention. This tool may have many potential appli-
cations in algorithms involving rooted vertex sets.

The algorithms constructed from the framework all have a very similar structure to them. A collapsing sweepcircle activates vertices. Once activated, a vertex begins to grow a region of influence (ROI). As a ROI grows, it collides with other regions of influence. When such a collision occurs, edges are added to the partial tree, at least one vertex deactivates, and the algorithm continues.

The flexibility of the algorithm framework lies in the use of independent modules to perform crucial tasks. The independent modules for the framework include the ROI definition and the component merge technique.

Three component merge techniques were suggested for use within the framework. The first of these was the spanning merge: when two regions collide, the edge between them is added, and the vertex furthest from the root deactivates. The second merge technique was the classic Steinerization merge: when two regions collide, the Toricelli (classic Steiner) point between these two vertices and the root is formed. Each vertex adds an edge to the Steiner point and deactivates. The final merge technique examined was the point of intersection Steinerization merge: when two regions collide, the point where these regions first meet becomes the Steiner point. Again, each vertex adds an edge to the Steiner point and deactivates.

Many ROI definitions can be used to produce arborescences. Two of these definitions have been examined in detail. Additional definitions are mentioned briefly in Section 7.2. Of course, the proper pairing of ROI definition and component merge techniques is critical to the resulting arborescence quality.

Utilizing the framework, four arborescence algorithms were created. The first two use a ROI definition consisting of a circle. Appropriately, these algorithms are called the circle spanning and circle Steiner arborescence algorithms, because they use the spanning and classic Steinerization component merge techniques, respectively. The other two algorithms have a ROI definition based upon lines tangent to the sweepcircle. The unrestricted tangent arborescence algorithm uses the classic Steinerization component merge technique. The restricted tangent arborescence algorithm uses an angle restricted version of the same ROI, and the point of inter-
Each of these algorithms was analyzed to determine its computational complexity and space requirements. Theoretical bounds for cost\(_{ratio}\), radius\(_{ratio}\) and radius\(_{max\, ratio}\) metrics were examined, although proofs of quality were not always forthcoming. The cost\(_{ratio}\) of an arborescence is the ratio of the arborescence cost to the minimum spanning tree (MST) cost. The radius\(_{ratio}\) of an arborescence is the radius of the arborescence divided by the radius of the shortest path tree (SPT). The radius of a vertex in an arborescence is the cost of the path between that vertex and the root. The final metric, the radius\(_{max\, ratio}\) is the largest of the ratios of the radius of a vertex and that vertex’s SPT distance to the root. A summary of the theoretical analysis results is presented in Table 7.1.

Finally, the quality of the arborescences generated by the algorithms were examined through empirical methods. The experiment was performed on large numbers of random point sets of various sizes. Three vertex distributions were examined: uniform, normal and clustered point sets. Analysis of the arborescences produced according to the three metrics outlined above was performed. This analysis provided insight into expected algorithm performance. Table 7.2 summarizes the experimental information gathered from each algorithm.

From a theoretical standpoint, ranking the algorithms is quite difficult. However, when experimental observations are examined, the situation is much clearer. The best overall algorithm is the circle Steiner arborescence algorithm. This algorithm

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Computational Complexity</th>
<th>Space Regmts</th>
<th>cost(_{ratio})</th>
<th>radius(_{ratio})</th>
<th>radius(_{max, ratio})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle Spanning</td>
<td>O((n^2 \log n))</td>
<td>O((n^2))</td>
<td>(\Omega(1 + \frac{4}{5}))</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Circle Steiner</td>
<td>O((n^2 \log n))</td>
<td>O((n^2))</td>
<td>(\Omega(\log n))</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Unrestricted Tangent</td>
<td>O((n \log n))</td>
<td>O((n))</td>
<td>(\Omega(\log n))</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Restricted Tangent</td>
<td>O((n \log n))</td>
<td>O((n))</td>
<td>open</td>
<td>(1 + \sqrt{2})</td>
<td>1 + \sqrt{2}</td>
</tr>
</tbody>
</table>

Table 7.1: Theoretical Algorithm Properties
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>(\text{cost}_{\text{ratio}})</th>
<th>Avg</th>
<th>Max</th>
<th>COV</th>
<th>(\text{radius}_{\text{ratio}})</th>
<th>Avg</th>
<th>Max</th>
<th>COV</th>
<th>(\text{radius}_{\text{max ratio}})</th>
<th>Avg</th>
<th>Max</th>
<th>COV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle Spanning</td>
<td>1.129</td>
<td>1.136</td>
<td>0.003</td>
<td></td>
<td>1.382</td>
<td>1.506</td>
<td>0.032</td>
<td></td>
<td>1.901</td>
<td>1.999</td>
<td>0.043</td>
<td></td>
</tr>
<tr>
<td>Circle Steiner</td>
<td>1.148</td>
<td>1.156</td>
<td>0.003</td>
<td></td>
<td>1.266</td>
<td>1.305</td>
<td>0.015</td>
<td></td>
<td>1.415</td>
<td>1.521</td>
<td>0.022</td>
<td></td>
</tr>
<tr>
<td>Unrestricted Tangent</td>
<td>3.120</td>
<td>3.204</td>
<td>0.004</td>
<td></td>
<td>1.853</td>
<td>1.876</td>
<td>0.005</td>
<td></td>
<td>1.868</td>
<td>1.887</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>Restricted Tangent</td>
<td>1.243</td>
<td>1.251</td>
<td>0.007</td>
<td></td>
<td>1.375</td>
<td>1.404</td>
<td>0.010</td>
<td></td>
<td>1.812</td>
<td>2.136</td>
<td>0.046</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.2: Experimentally Observed Algorithm Properties (5000 Vertices, Uniform Distribution)

had excellent performance under the three metrics examined. The downside to this algorithm is its computational complexity \(O(n^2 \log n)\) time) and space requirements \(O(n^2)\) space). The circle spanning arborescence algorithm also performed quite well on average, considering the nature of its merge technique (it does not use Steiner points). If the \(\text{cost}_{\text{ratio}}\) lower bound on worst case performance \(\Omega(1 + \frac{\pi}{6})\) had been realized earlier, perhaps the SPT bound could have been lowered, resulting in different experimental results (as discussed in Section 4.2.2.3). Regardless, their complexity and space requirements also make their use on large point sets much more expensive than the tangent arborescence algorithms.

Overall, the restricted tangent arborescence algorithm performed quite well. Experimentally, it was outperformed by the circle-based algorithms in almost all cases. However, this algorithm is less costly than the two circle-based algorithms, requiring only \(O(n \log n)\) time and \(O(n)\) space, thereby making it an excellent choice for use on large vertex sets. On the other hand, the unrestricted tangent arborescence did not perform very well under any of the metrics. Whereas the performance of the other algorithms with respect to a particular metric seemed to stabilize into a fairly constant rating as the number of vertices increased, the metrics for this algorithm continued to rise over the range of problem sizes analyzed. The problem seemed to be the ROI definition, which tended to move laterally, rather than in the direction of the root. When an intersection did occur, any Steiner points generated were gen-
erally inside the sweepcircle frontier which is almost always a poor choice for this ROI.

7.2 Future Work

Many possibilities for future research into arborescences arise directly as a consequence of this thesis. For example, the quality of the two circle arborescence algorithms show promise. However, their relative complexity \(O(n^2 \log n)\) make them too expensive to use on large point sets (in practice). Can ROI intersection events in the circle arborescence algorithms be determined more directly than the brute force techniques currently employed? In other words, can the complexity for the circle arborescence algorithms be reduced from \(O(n^2 \log n)\)?

One potential improvement may reduce the complexity of the circle ROI algorithms to \(O(n \log n)\) time. Rather than using brute force techniques to detect the next ROI intersection, use only those ROI intersection which can be easily detected. This can be accomplished by using the sweepcircle adjacency cycle to determine ROI intersection events. Only intersections between regions of influence that were previously adjacent on the sweepcircle will be detected. Conceptually, this may result in regions of influence colliding, but the collision not being detected. The arborescences may be somewhat different than they otherwise would be, but the complexity of the algorithm should drop to \(O(n \log n)\) time.

Another improvement involves the classic Steiner component merge technique. Currently, if the angles between the intersecting vertices and the root is greater than \(120^\circ\), then an edge from each vertex to the root is added, and each deactivated. This, however, may not be the best way to resolve this situation. Such a case can be seen in Figure 7.1(a), where vertices \(i\) and \(j\) have intersecting regions and vertex \(k\) lies just outside the ROI for vertex \(j\). The tree resulting from the current implementation is depicted in (b). However, simply rerouting vertex \(j\) to vertex \(k\) can reduce the cost, yet have little effect on the radius of the resulting tree.
Although performance guarantees have been shown for the arborescence radii, such proofs for cost\textit{ratio} have not been found. Can similar proofs be made for the cost\textit{ratio} metric? Additionally, is it possible to find theoretical average cost\textit{ratio} and radius\textit{ratio} values for each algorithm?

To further support the qualitative analysis made here, further investigation may involve alternative clustering methods. The clustering mechanisms employed during the empirical algorithm analysis could be improved upon. A suggested change would involve generating clusters with a Poisson distribution to simulate more natural clusters.

As well, the open-ended nature of the arborescence algorithm framework can potentially yield many more algorithms than those presented here. The flexibility of the framework is the result of the two independent framework modules: the ROI definition and the component merge technique.

The circle arborescence and tangent arborescence algorithms introduced two very different regions upon which to base algorithm progress. During the early stages of this research, other ROI definitions were considered. Many of those are still of interest:
• **Continuous River Basin ROI:** Recall that motivation for the circle arborescence algorithm came in part from the concept of the river Voronoi Diagram[23]. The concept was modified to represent water flowing into the drain of a basin. The ratio of boat movement to water movement was based upon a parameter $\sigma$. For the circle arborescence algorithm $\sigma = 1$ was examined. Further research could examine other specific values of $\sigma$, or examine it as a parameterized ROI, which could produce trees almost as inexpensive as the MST (when $\sigma = 0$), to trees as responsive as the SPT (when $\sigma = \infty$).

• **Angle Restricted Tangent ROI:** The region of influence for the restricted tangent arborescence had $\tau_2$ limited to be no more than 90°. It was thought that the choice of 90° would provide a reasonable amount of lateral expansion, yet still allow a SPT approximation bound. Given the quality and $O(n \log n)$ complexity of the restricted tangent (especially in comparison to the unrestricted tangent arborescence), examining the restricted tangent arborescence algorithm with a continuum of angle restriction values for $\tau_2$ may prove interesting.

• **External Circle ROI:** The region of influence for an active vertex is a circle whose diameter is the distance from the vertex to the sweepcircle frontier. (Example: Figure 7.2) With this ROI, pockets may form outside the sweepcircle frontier. This potential problem could be remedied by ignoring ROI intersections between vertices not adjacent in the sweepcircle adjacency cycle. The resulting algorithm could be compared to the two circle arborescence algorithms found in this thesis.

• **Concentric Circle ROI:** Another alternative region of influence is a growing circle centered at its defining vertex. The radius of the ROI for an active vertex $i$ is $\text{radius} = \overline{r} - \text{current.timestamp}$. (Example: Figure 7.3) The resulting algorithm may yield arborescences that compare favorably in cost to the MST.
The framework as discussed in Chapter 3 utilizes three different component merge techniques: spanning merge, classic Steiner merge and point of intersection Steiner merge. During early algorithm development one other Steiner component merge technique was considered. Consider Figure 7.5. Given that the regions of influence for two vertices \( i \) and \( j \) have intersected, locate the Steiner point as follows: Given that segment \( ir \) intersects the sweepcircle at \( a \), and segment \( jr \) intersects the sweepcircle at \( b \). Place the Steiner point \( t \) for the merge where segment \( ta \) intersects segment \( ja \). The relative advantages and disadvantages of such a Steiner point placement are not clear, but could be investigated.

![Figure 7.5: Alternate Merge Technique](image)

Alterations to the algorithm framework could be examined. Some of these would include sweepcircle variations. For example, if the sweepcircle started at the root and grew outwards (as opposed to collapsing), how would the regions of influence have to be modified? How would the resulting arborescences compare to those generated using the sweepcircle technique described here?

As well, could the sweepcircle be done away with altogether? As stated earlier, the sweepcircle places an artificial ordering to the point set, which should result in a faster algorithm (\( i.e. \), lower time complexity). However, if all vertices were activated at the start, how would the algorithm, the arborescences and the computational complexity change?
Alternatively, if vertices did not deactivate until their ROI intersected with the root, many more edges would be added. This would no longer constitute a tree, but graph-based arborescence techniques such as those discussed in Section 2.3 could be applied to this graph and may yield a quality arborescence.

An investigation of the framework under other distance metrics may also prove interesting. For example, the algorithms presented here may also work in the two dimensional plane under the $L_1$ distance metric. The regions of influence would change shape to reflect this metric change, as would the sweepcircle. But the arborescences produced could then be used in applications such as VLSI design.

Other possible avenues of research are further removed from the framework and algorithms presented here. For example, in regards to the concept of a peelable tree introduced earlier, what is the smallest spanning tree to harbor the peelable property? In other words, how difficult is it to find the minimum peelable spanning tree (MPST)? The peelable property in itself is not enough to make any guarantees with respect to radius (a spiral around the root may constitute the MPST yet have a virtually unbounded radius). However, can the LAST Tree techniques of Khuller, et al.[15] provide such guarantees when applied to the MPST?

Another technique for generating arborescences involves the weighted Voronoi Diagram. Recall that in a classic Voronoi Diagram, the plane is divided into $n$ regions, one for each vertex in the point set. The region for vertex $i$ represents all points which are closer to vertex $i$ than any other vertex. One technique for constructing Voronoi Diagrams involves expanding regions of influence (as discussed in relation to Figure 3.2). In the weighted generalization of the Voronoi Diagram, every vertex has associated with it a weight. This weight defines how fast the regions of influence grow. The result is circular shaped regions of influence when vertices have different weights, and the straight line Voronoi regions when the weights of two vertices are the same. Figure 7.6(a) depicts a weighted Voronoi region where the root $r$ has a weight of 2 and all other vertices have a weight of 1.

The dual of this weighted Voronoi Diagram is presented in Figure 7.6(b). From the resulting Euclidean graph, many spanning trees could be formed. One such
Figure 7.6: Weighted Voronoi Arborescences

tree (the shortest spanning tree on the Euclidean graph shown in (b)) appears in Figure 7.6(c). The power of this arborescence generating technique lies in the realization that when all vertices have the same weight, the DT (and therefore, the MST) can be easily generated. Alternatively, when the root has a weight which is arbitrarily large compared to the other vertices, the SPT is generated. The weights associated with the root and the other vertices can be manipulated, allowing the production of a spectrum of trees, varying in quality from the MST (with respect to cost) and to the SPT (with respect to radius).
References


