The Euclidean Steiner Tree Problem: Simulated Annealing and Other Heuristics

by

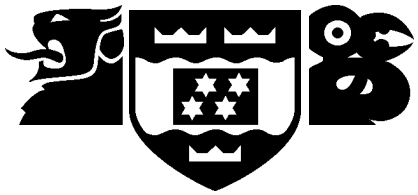
Geoffrey Ross Grimwood

A thesis

submitted to the Victoria University of Wellington
in partial fulfilment of the
requirements for the degree of

Master of Science
in Operations Research

Institute of Statistics and Operations Research
VICTORIA UNIVERSITY OF WELLINGTON



1994

Abstract

In this thesis the Euclidean Steiner tree problem and the optimisation technique called simulated annealing are studied. In particular, there is an investigation of whether simulated annealing is a viable solution method for the problem. The Euclidean Steiner tree problem is a topological network design problem and is relevant to the design of communication, transportation and distribution networks. The problem is to find the shortest connection of a set of points in the Euclidean plane. Simulated annealing is a generally applicable method of finding solutions of combinatorial optimisation problems. The results of the investigation are very satisfactory. The quality of simulated annealing solutions compare favourably with those of the best known tailored heuristic method for the Euclidean Steiner tree problem.

Acknowledgements

An enormously large thank you to my partner Carol. My return to university would not have been possible without Carol's support and encouragement.

I am grateful to my supervisor Dr Tapas Sarkar. Firstly, for his invaluable guidance and advice. Secondly, for pointing me in the direction of the Steiner problem and suggesting simulated annealing as a possible solution method.

Thanks also to Dr Rod Downey of Victoria's Mathematics Department for assisting me in vastly improving my understanding of computational complexity.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Outline

In this thesis a particular problem and a general method of solving problems are brought together. The problem is the Euclidean Steiner tree problem and the method of solution is simulated annealing. The objective of this thesis is to investigate whether simulated annealing can give solutions of comparable quality to those produced by existing purpose built, or tailored, heuristics for the Euclidean Steiner tree problem. Achieving this goal requires an understanding of the problem, existing heuristics for the problem and the general simulated annealing approach to problem solving.

The Euclidean Steiner tree problem is one of finding the shortest connection of a set of points in the Euclidean plane (see Hwang *et al.* [19]). It is more than just finding a simple minimum spanning tree because it is possible to introduce additional points that help to give a shorter connection. The additional points are called Steiner points. At first the problem appears to be purely geometrical in nature. There is a substantial amount of geometry involved but it is also a combinatorial problem. One of a finite number of possible configurations is the shortest connection. The problem has been shown to be very difficult and is highly unlikely to have an efficient method of solution. It is for this reason that much time and effort is spent on developing approximate methods with the desirable property of generally finding solutions close to the best.

The problem is one of many that belong in the domain of network design.[1] Other problems in this area are the familiar minimum spanning tree and travelling salesman problems. Network design is comprised of three main areas, topological, routing and capacitated design. These aspects are interrelated, a change to or evaluation of a solution to one must be considered in light of effects on the other two components. It is often necessary to reassess the design problems in such circumstances.

- Topological design is concerned with the number and location of sites and the interconnection of sites.
- Routing design involves the scheduling and routing of items of interest across a network.

[1] See Smith and Winter [36] for a discussion of network design. In particular topological network design from a computational geometry perspective.

- Capacitated design is concerned with determining the best size of sites and interconnections to cater for flows across a network.

The Euclidean Steiner tree problem is an example of a topological design problem, as are the minimum spanning tree and travelling salesman problems.

Simulated annealing is a method of solution often applied to difficult combinatorial optimisation problems. It is based on the analogy with the physical process of annealing. In annealing a substance is heated to melting point and then allowed to cool very slowly. The substance will eventually be in a state of low energy. This process can be simulated on a computer for many particle systems to find minimum energy configurations (see Metropolis *et al.* [29]). The analogy with combinatorial optimisation is: possible solutions to a combinatorial problem are likened to the states of the substance, and many small random changes are made to a solution. As the "temperature" decreases fewer and fewer increasing cost changes are accepted. The final "frozen" solution is the globally optimal solution to the combinatorial problem (see Kirkpatrick *et al.* [23], Černy [4], Laarhoven [25]).

Simulated annealing has been applied to many problems with mixed success, in the sense that tailored heuristics generally outperform annealing in terms of quality of solution and always in terms of computational effort. However, it is annealing's general nature that makes it possible to apply it to problems that do not have efficient optimal algorithms or tailored heuristics. Or to variations on problems where the "parent" problem has a tailored heuristic, but which is utterly useless for solving the variation.

In this thesis there is this chapter and six others. Two chapters can be read independently of all others, Chapters 2 and 5. The first is an introduction to the Euclidean Steiner tree problem, the second is an introduction to simulated annealing. There are two threads of chapters beginning with these two chapters and coming together at Chapter 7 "Simulated Annealing and the Euclidean Steiner Tree Problem". Figure 1.1 shows the organisation and dependencies of chapters. If the reader is primarily interested in the Euclidean Steiner tree problem *and* simulated annealing then Chapters 3 and 6 can be omitted. Nevertheless, both are valuable in the sense of gaining an appreciation of the Steiner problem and simulated annealing respectively.

Chapter 2 is an introduction to the Euclidean Steiner tree problem. The terminology and notation associated with the problem are defined and examples given. Topologies are fundamental to describing possible solutions to the problem. In particular, Steiner topologies are defined and an expression is given for the total number of ways of connecting a set of points. The shortest connection is one of many Steiner topologies. An important result is that a Steiner topology can be decomposed into a union of full Steiner topologies. This result is at the heart of nearly all optimal algorithms for the problem. The methods for constructing full Steiner trees for given full topologies are described and shown by example. Also described is the Steiner polygon. This is used to decompose a problem into a number of smaller subproblems. Finally, optimal algorithms are outlined, in particular the first algorithm to be implemented by Cockayne and others (see [19]). In this chapter there is a section on computational com-

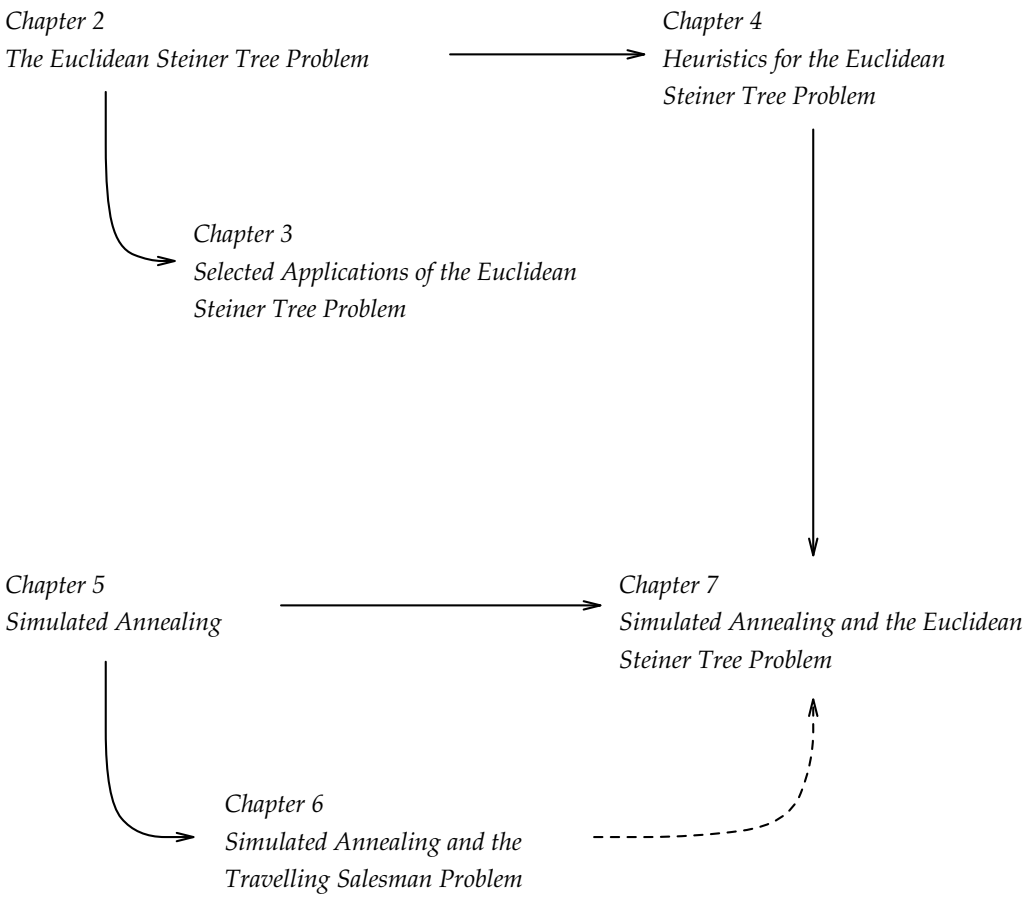


Figure 1.1. Organisation and dependencies of chapters in this thesis.

plexity. The classes NP, NP-complete and NP-hard are informally defined and their importance explained. The complexity of the Euclidean Steiner tree problem is described. This introduction within an introduction is needed so that it is possible to describe the Steiner problem's complexity and show that the problem belongs to the class of most difficult problems currently known.

The third chapter is a collection of descriptions and examples of three generalisations of the Steiner problem. It demonstrates that the problem is a fundamental problem of network design, and that most network design problems are very difficult. The first problem is one of optimally connecting new points to an existing network, for example new customers to an existing telephone network. The second is to find the minimum cost network for a communication systems where cost is dependent on capacity to be provided on links making up the network and length of links. Steiner trees give good solutions in many cases although not necessarily the best. The final problem described is the design and routing of building services, for example heating and ventilation, and plumbing.

Tailored heuristics for the Euclidean Steiner tree problem are the subject of Chapter 4. A brief survey of heuristics is given. Two heuristics are described in detail. The first is by Smith *et al.* [37] and uses the geometrical structures called Voronoi polygons and Delaunay triangulations to find approximate shortest connections. The heuristic is described by example and suggestions are made for possible improvements, or at least changes to be investigated in the hope of giving gains in quality of solution. The second heuristic is by Beasley and Goffinet [2]. It also uses Voronoi polygons and Delaunay triangulations. Their experience shows it to be the best known heuristic for the problem. An example of the heuristic is given in an appendix.

Chapter 5 is an introduction to simulated annealing and is based on the work of Laarhoven [25]. The advantages and disadvantages of tailored heuristics for combinatorial optimisation problems are described, and compared to those of the simulated annealing approach. The physical process of annealing, the Metropolis algorithm and the analogy with combinatorial optimisation which gives simulated annealing are discussed. A mathematical model of annealing using Markov chains is outlined, and used to define conditions for the asymptotic convergence of the annealing algorithm to a globally optimal solution. Fundamental to the operation of simulated annealing is the cooling schedule, this controls the behaviour of the "temperature", called the control parameter. General considerations for determining a starting value, change in and stopping value for the control parameter are described. In a real algorithm asymptotic convergence is not possible in time bounded by a polynomial in the problem size. However it is possible to approximate the convergence using what is known as a polynomial cooling schedule (Laarhoven and Aarts [24], [25]). The polynomial schedule is described. Finally, modifications to the standard simulated annealing algorithm are briefly discussed.

The application of simulated annealing to the travelling salesman problem is investigated in Chapter 6. Much work has been done on using annealing to solve this archetypical combinatorial problem. This chapter provides a means of becoming familiar with annealing and investigating the quality of solutions

and computation time as functions of the polynomial cooling schedule parameters. Annealing is applied to random problems of varying size and to test problems found in the literature which have known optimal solutions. The results show that the quality and time are primarily dependent on one parameter of the schedule. The comparison of annealing with the k-opt tailored heuristic confirms the work of Laarhoven: simulated annealing can not compete with tailored heuristics for the travelling salesman problem and in general for many other problems.

Chapter 7 is the chapter in which the Euclidean Steiner tree problem and simulated annealing threads are brought together to investigate whether simulated annealing can compete with tailored heuristics, and in particular the new benchmark heuristic of Beasley and Goffinet. The scarcity of references on the Euclidean Steiner tree problem *and* simulated annealing, and the quality of the tailored heuristics momentarily invokes a feeling of despair. However, the exercise remains. A very simple annealing transition mechanism is developed. Random additions, deletions and replacements of points in a solution, comprising a collection of Steiner points, are described and demonstrated. The results when applied to randomly generated problems of up to 100 points indicate good quality solutions can be obtained within one to one and a half hours of CPU time using a particular set of annealing parameters. The annealing is also applied to Cockayne and Hewgill's thirty 100 point test problems with known optimal solutions. Beasley and Goffinet give results for their heuristic for the same set of problems. The comparison is encouraging. Finally, a simple local improvement procedure is applied to the annealing solutions for the one hundred point problems and the comparison with Beasley and Goffinet is repeated. The differences in quality of solutions are negligible. The initial despair is replaced by mild excitement when it is shown that simulated annealing (admittedly with some local improvement) can compete with a tailored heuristic in terms of quality of solution.

In this thesis there are also a number of appendices containing listings of the Steiner polygon, travelling salesman and Steiner tree annealing programs, Euclidean Steiner tree test problems and travelling salesman test problems, an example of Beasley and Goffinet's heuristic and examples of an annealing transition mechanism for the travelling salesman problem.

Chapter 2

The Euclidean Steiner Tree Problem

In this chapter the Euclidean Steiner tree problem is introduced. It is a relatively lengthy chapter as much is covered. It begins with a brief history of what is an old problem of mathematics. The variants of the problem and applications are briefly introduced. This is followed by a section on computational complexity, which concludes with a discussion of the Euclidean Steiner tree problem's complexity. The basic definitions and properties of Steiner trees and their underlying topology are described. The fundamental geometrical objects called Steiner polygons and full Steiner trees are defined and examples of constructing them are shown. The final section is a description of optimal algorithms developed over the last twenty or so years to solve the Euclidean Steiner tree problem.[1]

2.1 A Little History

The origins of the Euclidean Steiner tree problem date back to Fermat's problem in the 17^{th} century: given three points find a fourth point that minimises the sum of the distances from each of the three points to the fourth. Torricelli (before 1640) showed that the intersection of the three circles circumscribing the equilateral triangles formed by each side of the triangle made by the three points is the sought after point and it is called the *Torricelli point* (if all the angles in the triangle are less than 120°). Cavalieri (1647) showed that the three lines from the three given points to the Torricelli point make an angle of 120° with each other at the Torricelli point. Simpson (1750) proved that the lines joining the outside points of each equilateral triangle to the opposite vertex of the given triangle intersect at the Torricelli point. The three lines are called the *Simpson lines*. Heinen (1834) proved that the lengths of the Simpson lines are identical and equal to the sum of the lengths of the lines from the given points to the Torricelli point. Many others studied Fermat's problem including the

[1]The two main sources for this chapter are Winter [42] and Hwang *et al.* [19] (to a lesser degree). Two comprehensive surveys exist on the Steiner problems: a paper by Hwang and Richards [17] and a book by Hwang *et al.* [19]. Both cover the Euclidean Steiner tree problem and two other important variants of the Steiner problem, the network (or graphical) and rectilinear problems.

Swiss mathematician Jacob Steiner in the 19^{th} century. Figure 2.1 shows the numerous geometrical objects.

It was not until 1934 that what is now known as the Euclidean Steiner tree problem appeared in a paper by Jarnik and Kössler.[2] Their problem was: find a shortest connection of n points in the plane. In 1941 Courant and Robbins [8] discussed the shortest connection problem and, perhaps mistakenly, called the problem the Steiner problem.[3] Their problem was: *given a set of points in the Euclidean plane find a shortest connection where additional points may be introduced as junctions of connecting lines if introduction of the additional points gives a shorter length connection*. This is the Euclidean Steiner tree problem. The additional points are called *Steiner points*.

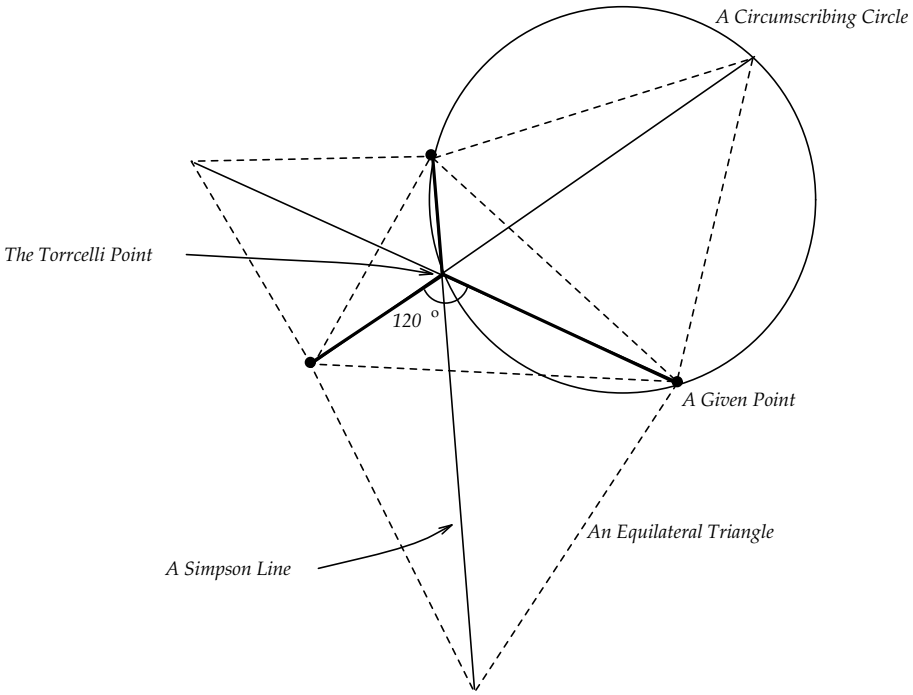


Figure 2.1. The numerous geometrical objects defined to aid solving Fermat's problem.

In 1961 Melzak [28] was the first to show that the shortest connection could be found in a finite number of steps and also established many of the basic properties of the shortest connection. Since then many researchers have given the Steiner problem a great deal of attention and a vast literature exists on the Euclidean Steiner tree problem and other variants of the Steiner problem.

[2]Hwang *et al.* [19] provide a reference. It is a foreign language journal, possibly Czech or Slovak.

[3]Courant and Robbins make no mention of Fermat or Jarnik and Kössler. Further, Steiner was one of many to delve into Fermat's problem and perhaps deserved the honour of having the problem named after him no more than other researchers. But there is no point in worrying about it now.

2.2 Applications

In this section applications of the Euclidean Steiner tree problem and other variants of the Steiner problem are described. In Chapter 3 three applications of the Euclidean Steiner tree problem are described in greater detail.

An obvious application of the Euclidean Steiner tree problem is to finding the shortest routing of roads, telecommunication links or pipelines. Generalisations of the problem provide a greater variety of possible applications at the expense of added complexity. For example, adding obstacles provides greater realism, but also tremendously complicates matters. An example would be the design of a road system in the presence of mountain ranges and lakes. A further application is the routing of services in buildings where possible routes are limited. The inclusion of costs on edges gives the opportunity of analysing drainage or pipelines networks where the effect of gravity is an important consideration.

A variant of the Steiner problem is the network Steiner problem. The problem is to find the shortest network connecting a subset of vertices of a graph where vertices not in the subset, if any, can be used if a shorter connecting network results. Many network design problems are generalisations of the network Steiner problem.[4]

Another variant is the rectilinear Steiner problem. The rectilinear problem is in the Euclidean plane but the distance metric is the Manhattan metric not the Euclidean metric. The distance from point a to b is given by $|x_a - x_b| + |y_a - y_b|$. The network problem is a generalisation of the rectilinear problem. An algorithm to solve the graphical problem can also be used to solve the rectilinear problem. However the special geometry and importance of the rectilinear problem has lead to it being extensively studied and being considered a problem in its own right. The connection with the graphical problem is that the rectilinear Steiner tree is a subgraph of the grid graph induced by the points. The grid graph is formed by drawing horizontal and vertical lines through each given point. The intersections of these lines (other than at the given points) are vertices that can be used to give a shorter length connection. The rectilinear problem is applied to circuit layout on computer chips and boards and the routing of services in a building.

The final application is from the world of biology. An area of interest to biologists is the construction of evolutionary trees or phylogenetic trees. The objective is to take existing species and build a tree joining the species to hypothetical ancestors. A possible distance measure is the number of likely steps to evolve from one species to another. The reason for attempting to construct such a tree may be either to find the relationships between species or to measure the difference or similarity between species. Steiner trees are an appropriate mathematical model for these evolutionary trees. Unfortunately defining a suitable point space and metric for measuring distance within the space are cause of much debate.

[4] A network is a collection of points or *vertices* and a collection of *edges* joining the vertices. A network connecting n vertices using exactly $n - 1$ edges is called a *tree*.

2.3 Computational Complexity

Computational complexity provides a means of calibrating the difficulty of problems in terms of various resources. If a problem can be shown to be *inherently "intractable"* then an exact method of solving the problem will generally be of little practical use. Attention is best focused on finding approximate solutions, or relaxing the conditions of a problem and searching for a solution that meets most of the problem requirements. In this section computational complexity is informally introduced. This is followed by a discussion of the complexity of the Euclidean Steiner tree problem.[5]

2.3.1 Preliminary definitions

A *problem* is a question for which an answer is desired, and is generally stated as a list of parameters defining the problem and a set of conditions that an answer or *solution* to the problem must satisfy. For example, the Euclidean Steiner tree problem can be stated as: given a set of points $\mathcal{A} = \{a_1, \ldots, a_n\}$ in the plane, what is the set of points $\mathcal{S} = \{s_1, \ldots, s_k\}$, $k \geq 0$, that gives the minimal length spanning tree of $\mathcal{A} \cup \mathcal{S}$? An *instance* of the problem is a particular set of points $\mathcal{A}$. For example an instance of the Steiner problem is $\mathcal{A} = \{(4,7),\ (6,9),\ (9,6)\}$. An *algorithm* is a step-by-step procedure for finding a solution to a problem. An algorithm *solves* a problem if for any instance it produces a solution. The solution to the example instance is $\mathcal{S} = \{(5.9458, 7.9885)\}$.

The *efficiency* of an algorithm is measured by the resources required to solve a problem. Time is the usual resource measured, although memory is sometimes of interest. The most efficient algorithm is generally the fastest algorithm. This requirement is expressed as a function of the *size* of a problem. The size is the length of the input to the algorithm.[6] The *time complexity* of an algorithm is a function from the input size to the *maximum* running time of the algorithm on instances of that size.[7]

The time complexity function provides a simple but extremely powerful classification of algorithms and problems. An algorithm with time complexity function $f(n)$ is said to be a *polynomial algorithm* if $f(n)$ is $O(p(n))$ where $p(n)$ is a polynomial function of n.[8] If $f(n)$ can not be bounded by a polynomial then the algorithm is said to be *super-polynomial*. A typical super-polynomial time is $O(c^n)$ with $c > 1$ and constant, and is called *exponential*. Most practical algorithms seem to be either polynomial or exponential.

[5] Garey and Johnson [13] is an excellent introduction to the theory of computational complexity and is the source of this informal discussion. The discussion uses terms such as problem, algorithm and instance. The formal theory uses languages, Turing machines and strings.

[6] The formal theory measures size as the length of an instance expressed as string of symbols. An *encoding scheme* transforms a problem instance into a string.

[7] The maximum or worst case is used because different instances of the same size can have different running times. Tests on input or the results of earlier steps in an algorithm may lead to different steps being performed, hence different running times.

[8] $f(n)$ is $O(g(n))$ if there exists a positive constant c such that $|f(n)| \leq c|g(n)|$ for all $n \geq 0$. The property of polynomials of importance in computational complexity is the closure property. If $f(n)$ and $g(n)$ are polynomial then so too are $f(n) + g(n)$, $f(n) \times g(n)$ and $f(g(n))$.

This scheme also allows classification of the time complexity of problems. A problem is polynomial if some polynomial algorithm exists to solve the problem. Problems can be divided into *tractable* and *intractable* classes in the sense that polynomial is tractable and super-polynomial is intractable. It seems most intractable problems are exponential or worse. It is possible to provide examples of time complexity functions that upset the classification, for example an "efficient" algorithm with polynomial time complexity function n^5 takes more time than the "inefficient" algorithm with exponential function 2^n for $n \leq 20$. However, experience has shown that for many optimisation problems exponential algorithms are of little use for large input sizes, and researchers continue to devote time to finding polynomial algorithms. Of course, algorithms with polynomial functions such as 64^7n^2 or $3n^{205}$ are of little practical use. But again experience has shown that in general for "naturally" occurring polynomial problems the polynomials are of low order with modest coefficients.

2.3.2 Intractability

The notion of intractability is independent of the how the input to an algorithm is expressed and is independent of the model of computation used. In slightly more formal terms, the size of the inputs under different methods of reasonable coding, for example binary and decimal, are only polynomially different, and therefore an algorithm solving a problem in polynomial time under one coding will still be polynomial under any other reasonable coding. A reasonable computer model is one where there is a polynomial bound on the amount of work that can be done in unit time. Therefore an algorithm with polynomial time under one model will have polynomial time under any other reasonable model.

Some problems are provably intractable in the sense that *any* algorithm *must* be super-polynomial, for example tiling problems and some problems in logic. But what can be said about problems that are neither provably intractable nor *currently* possess a polynomial algorithm? *Are they inherently intractable?*[9]

2.3.3 NP-completeness

In this section the key notion of NP-completeness is introduced, and is discussed in terms of *decision problems*. A decision problem has a "yes" or "no" solution. This, however, does not mean the theory is solely concerned with decision problems, optimisation problems can be discussed because every optimisation problem has a related decision problem. For example, the travelling salesman decision problem is: given a set of cities and the distances between every pair of cities *is there a tour with length less than some number* K visiting each city once and only once and returning to the starting city? The decision

[9]There are some problems which are intractable because the size of their solution can not be bounded by a polynomial, for example list all tours of a travelling salesman problem with length less than some number K. It is possible to construct an instance with exponentially many tours satisfying the condition. Solutions to problems of this type and therefore the problems themselves are of little practical value because of the sheer size of the solutions.

problem can be used to solve the optimisation problem. For example, let TSP be the travelling salesman optimisation problem and $TSP(K)$ be the decision version defined above. By asking $TSP(1), TSP(2), \ldots$ until a "yes" solution is found the TSP problem can be solved. This is an example of a reduction and is discussed below. The reduction imposes a partial ordering on problems, in the sense of a problem being as hard as another, and allows problems to be calibrated according to their time complexity into classes. For example, TSP is at least as hard as $TSP(K)$.

Important classes are the class P, short for polynomial, and EXP, exponential. A problem is in P if it can be solved in polynomial time using a reasonable model of computation. NP, short for non-deterministic polynomial, contains problems for which a proposed solution can be checked in polynomial time. This is not solving the problem in polynomial time, it is simply checking a proposed solution is correct in polynomial time. For example, given a tour for the travelling salesman problem it is easy to check it is a valid tour and compute its length. It is obvious that a problem in P is in NP. The checking can be replaced by actual solution of the problem, and because this can be done in polynomial time the problem is in NP. This implies P $\subseteq$ NP. What is not known is whether P $=$ NP. This is a fundamental open question of mathematics and computer science. It is currently believed P $\neq$ NP and it is the difference between P and NP that is of importance. *Given a problem in NP and assuming P $\neq$ NP is the problem in P or NP $-$ P?* That is, is the ability to check a proposed solution the same as finding a solution? It is conjectured that if the problem is in NP $-$ P then it is intractable.

A further important class is NP-complete. The NP-complete class is such that if one problem in the class can be proved to have a polynomial algorithm then all NP-complete problems have a polynomial algorithm. And conversely, if one problem can be shown to be intractable then all are intractable. The equivalence of NP-complete problems is established through *reductions* of problems to other problems. For instance, many combinatorial optimisation problems can be reduced to general zero-one linear programming problems.

Polynomial reducibility or *transformation* of one problem to another is fundamental to NP-completeness. A polynomial transformation of a problem instance is a mapping from the coding scheme of one problem to another and can be performed in polynomial time. This means that if problem Y has a polynomial algorithm and there is a polynomial transformation from problem X to problem Y then X has a polynomial algorithm. This can be thought of as meaning X *is at least as hard as* Y. Polynomial transformations can be used to make statements about problems, for example if $X \in$ P and Y can be transformed to X then $Y \in$ P. Further, equivalence classes exist based on transformations, P is one and NP-complete is another. A problem X is NP-complete if $X \in$ NP and for all other $Y \in$ NP, Y can be polynomially reduced to X. Therefore all NP-complete problems are as hard as each other and are the hardest in NP. The important, although conditional, property of an NP-complete problem X is that $X \in$ P if and only if P $=$ NP.

To show a problem is NP-complete does not necessarily involve showing that every problem in NP can be polynomial transformed to it, the equivalence

relation means it is only necessary to show the problem is in NP and *one* NP-complete problem can be transformed to it. The number of NP-complete problems has grown rapidly since the first NP-complete problem was proven to be so. This provides an "easy" method of determining or verifying the apparent intractability of a problem: *Try to prove it is NP-complete!* If it can not be done, perhaps because the problem can not be shown to be in NP, then all is not lost, it is still possible to make "at least as hard as" statements.

2.3.4 NP-hardness

Many problems are not in NP. Optimisation problems and some decision problems are not in NP. What can be said about them? Decision problems that can be transformed to a known NP-complete problem are said to be at least as hard as NP-complete problems and are described as being *NP-hard*.[10] Optimisation problems with corresponding decision problems in NP-complete are also NP-hard. Figure 2.2 shows the relationships between the classes of problems.

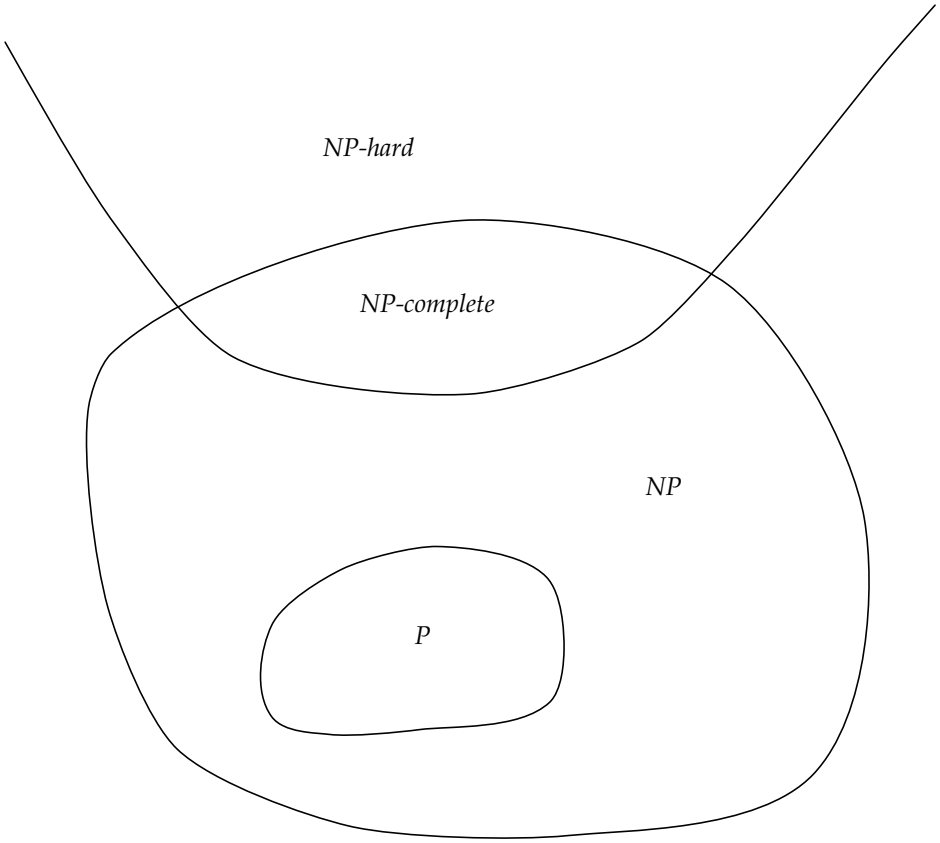


Figure 2.2. The classification of problems according to complexity.

[10] All NP-complete problems are NP-hard.

2.3.5 The Euclidean Steiner tree problem's complexity

Garey *et al.* [14] show that Euclidean Steiner tree problem's related decision problem *is at least as hard* as any NP-complete problem. But are unable to show the decision problem is in NP because of difficulties with irrational numbers.[11] The difficulties are with computers' inabilities to represent and manipulate irrational numbers because only a finite amount of storage exists. The Euclidean Steiner tree problem involves irrational numbers: Steiner points can be at any location in the plane and may be irrational, and the Euclidean metric can give irrational lengths and therefore make comparison of lengths difficult.

To better match the real world Garey *et al.* formulate a related discrete optimisation problem. In this problem the set of points to be connected and any Steiner points are required to have integer coordinates. And the normal Euclidean measure is replaced by the discrete Euclidean measure: the length of the edge joining points a and b is given by $\left\lceil \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2} \right\rceil$, where $\lceil x \rceil$ is the smallest integer greater than or equal to x. This discrete Euclidean Steiner tree problem can be made to approximate the original problem to any degree of accuracy by suitable scaling of coordinates. The decision problem of the discrete optimisation problem is:

> Given a set of points with integer coordinates and an integer L is there a Steiner tree with discrete Euclidean length less than L, where Steiner points, if any, have integer coordinates?

Garey *et al.* show this decision problem is NP-complete. They also demonstrate that the Euclidean Steiner tree problem is at least as hard as the discrete problem by showing that no polynomial time algorithm which outputs symbolic expressions involving $+$, $-$, $\times$, $\div$, $\sqrt{}$ for Steiner points exists to solve the Euclidean Steiner tree problem.[12]

The Euclidean Steiner tree problem's decision problem is NP-hard but not in NP, and therefore is not NP-complete. Although limited to "a least as hard as" statements it seems reasonable to say the Euclidean Steiner tree problem is an extremely difficult optimisation problem, perhaps more so than optimisation problems whose decision problems are NP-complete, for example the travelling salesman problem and the other variants of the Steiner problem. This intractability becomes evident in later sections when the the number of possible candidates for the shortest connection is determined. It grows very, very rapidly with increasing problem size.

2.4 Basic Definitions

The problem of finding the shortest network connecting a set of points in the Euclidean plane is called the *Euclidean Steiner tree problem*. A formal definition of

[11] Both the Network and Rectilinear Steiner tree problems' related decision problems are NP-complete.

[12] If a polynomial algorithm does exist then all NP-complete problems have a polynomial time algorithm.

the problem is: given a set of n points in the Euclidean plane $\mathcal{A} = \{a_1, \ldots, a_n\}$ find the shortest connecting network where it is possible to introduce other points as junctions of edges. Points in $\mathcal{A}$ are called *A-points*, junctions in the network that are not in $\mathcal{A}$ are called *Steiner points* or *S-points*. The shortest connecting network is called the *Steiner minimal tree* or *SMT*.

A shortest connecting network with no S-points is called a *minimum spanning tree* or *MST*. All edges in a MST connect A-points only. The MST is important because its length provides an upper bound on the length of the SMT and it can be used in the search for the SMT. Finding a MST is straightforward and can be done quickly.

Figure 2.3 shows the SMT for a set of eight points. This SMT has four S-points. The MST for the same set of eight points is shown in Figure 2.4. In this example the SMT is about 5% shorter than the MST.

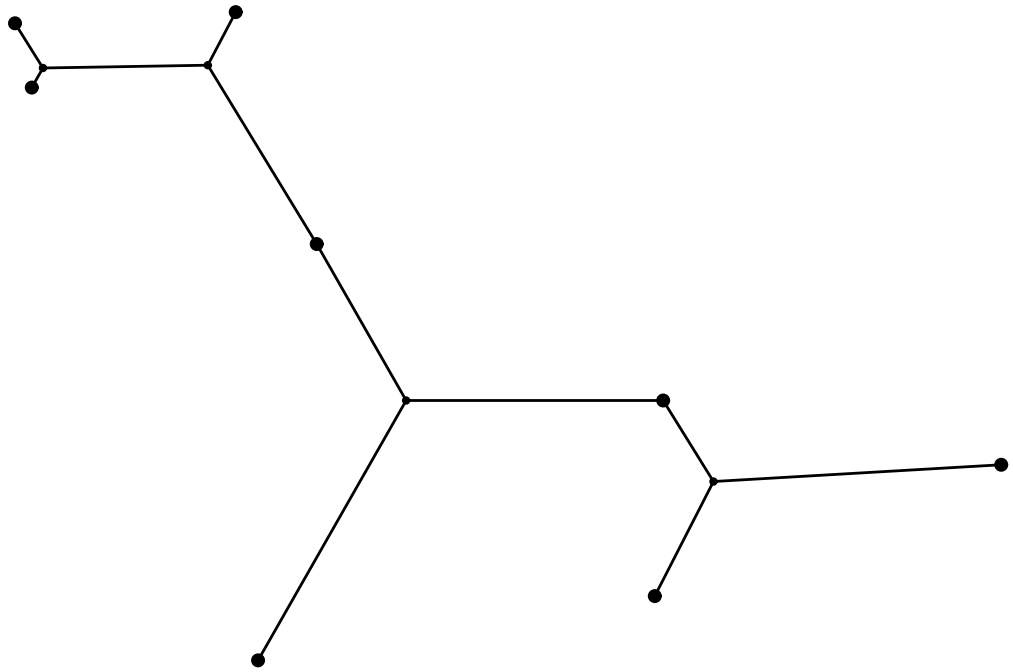


Figure 2.3. The Steiner minimal tree for a set of eight points.

The structure of a network or the *topology* can be shown graphically by another network with the same vertices and edges but with all edges at 90° or 180° to each other. Figure 2.5 shows a six point network on the left and the graphical representation of the network's topology on the right. For a given topology the shortest connecting network is called the *relatively minimal tree*. The connection is minimal relative to the given topology.

Each vertex in a network has a *degree*. The degree is the number of edges incident on the vertex. For example the degree of vertex A in Figure 2.5 is one and the degree of D is two.

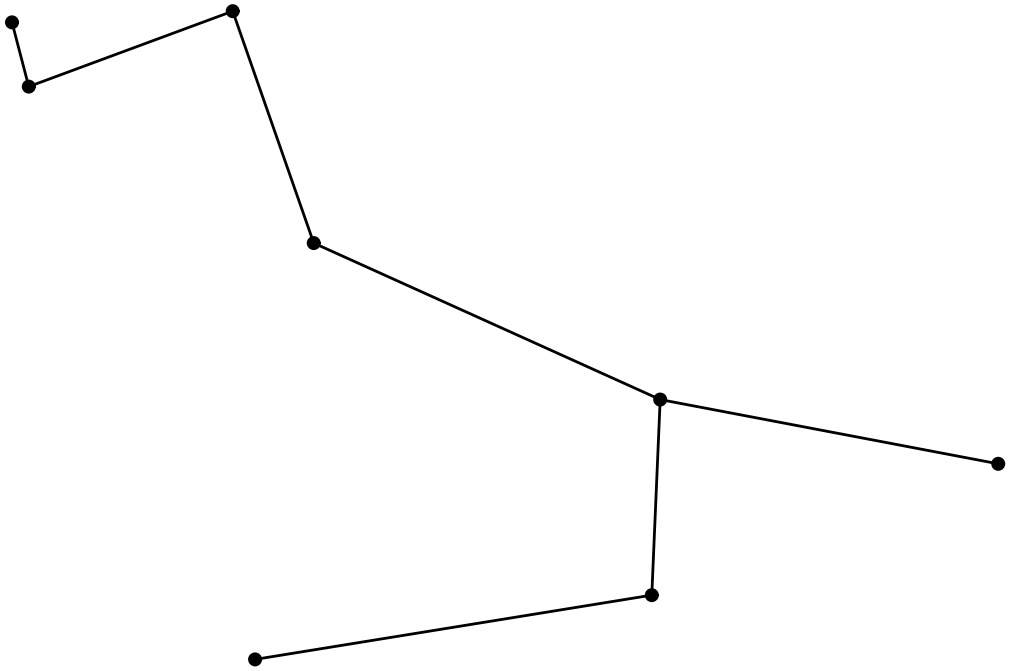


Figure 2.4. The minimum spanning tree for a set of eight points.

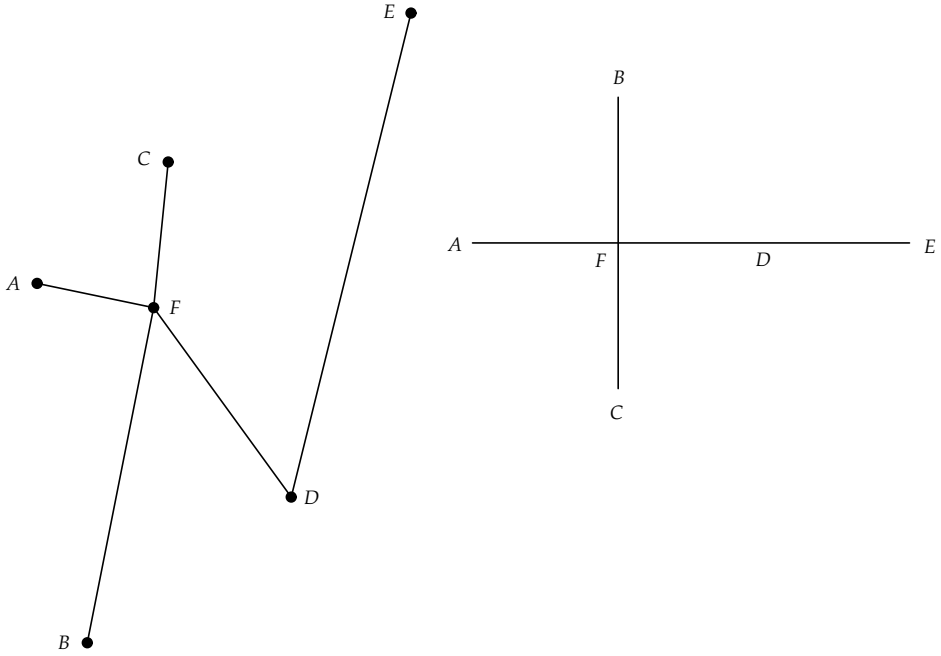


Figure 2.5. An example of a tree network and a graphical representation of its topology.

2.5 Basic Properties of Steiner Minimal Trees

Some of the properties of a Steiner minimal tree are:

- The length of the SMT of a set of points $\mathcal{A}$ is less than or equal to the length of the MST of $\mathcal{A}$;
- The length of the SMT is at least $\frac{\sqrt{3}}{2}$ times the length of the MST;[13]
- In a SMT no pair of edges meet at an angle less than 120°;
- The degree of every S-point in a SMT is exactly three. Therefore edges incident on an S-point make an angle of exactly 120° with each other;
- The number of S-points, k, in a SMT is at most $n-2$;
- Every A-point has a degree of at most three;
- If there are $k = n-2$ S-points in a SMT then the degree of every A-point is one;
- If $a_i a_j$ is an edge in a SMT, where both a_i and a_j are A-points, then $a_i a_j$ *must* be an edge in the MST of the set of A-points;
- No edges of a SMT other than edge $v_i v_j$ pass through $L(v_i, v_j)$, where $L(v_i, v_j)$ is the lune shape defined by the the intersection of the two circles of radius $|v_i v_j|$ centered at v_i and v_j. This property is illustrated in Figure 2.6;
- A wedge defined by three A-points with an angle greater than 120° and containing no other A-points contains no S-points and no part of a SMT lies in the wedge. In Figure 2.7 an SMT is shown together with two wedges that have angles greater than or equal to 120° and contain no other A-points. Neither wedge contains any S-points.

The lune and wedge properties described above are fundamental to the Steiner polygon defined in Section 2.8.

2.6 Steiner Topologies

Topologies satisfying the following three conditions are called *Steiner topologies*:

- the number of S-points is at most $n-2$;
- every A-point has a degree of at most three;
- all S-points have a degree of exactly three.

[13] $\inf_{\mathcal{A}} \frac{|SMT(\mathcal{A})|}{|MST(\mathcal{A})|} = \frac{\sqrt{3}}{2} \approx 0.866$ is known as the *Steiner ratio* and is much studied (see Du and Hwang [10]). It means that the SMT is never more than about 13% shorter than the MST. The Steiner ratio and the MST provide measures for comparison of heuristic methods of computing a SMT.

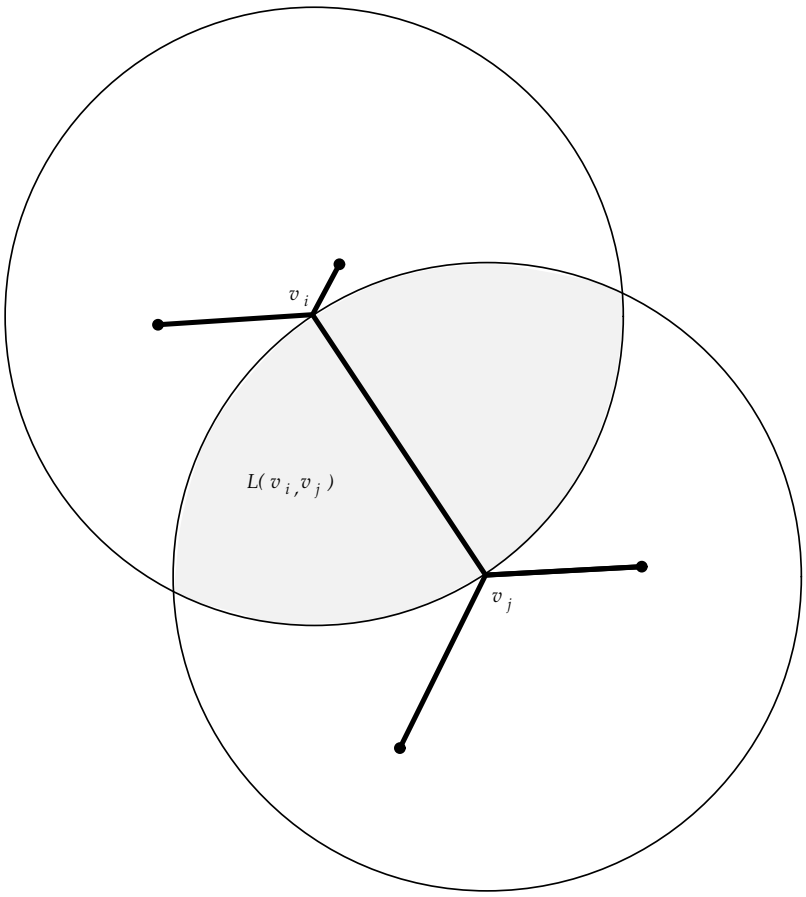


Figure 2.6. An example of the lune property of a Steiner minimal tree. The only edge to pass through the lune $L(v_i, v_j)$ is edge $v_i v_j$.

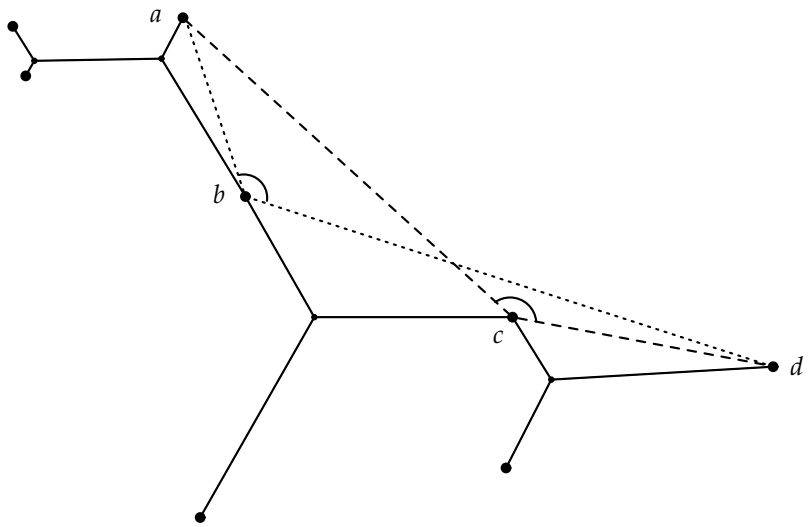


Figure 2.7. An example of the wedge property of a Steiner minimal tree. A-points a, b and d form a wedge with angle abd not less than 120° and contain no other A-points. Similarly for the wedge defined by a, c and d. Neither wedge contains any S-points and no part of the SMT lies in either wedge.

An important subset of the set of Steiner topologies is the set of *full Steiner topologies*. A full Steiner topology for n A-points has exactly $n-2$ S-points. An expression for the number full Steiner topologies is

$$f(n) = \frac{2^{-(n-2)}(2n-4)!}{(n-2)!}.$$

Table 2.1 shows the number of full Steiner topologies for different values of n. The function $f(n)$ grows rapidly. It will be all too obvious in later sections that this extremely fast growth is the bane of optimal algorithms for the Euclidean Steiner tree problem.

The number of Steiner topologies with n A-points and $k \leq n-2$ S-points where no A-point is of degree 3 is denoted by $F(n,k)$, and given by

$$F(n,k) = \left(\begin{array}{c} n \\ k+2 \end{array} \right) f(k) \frac{(n+k-2)!}{(2k)!}.$$

This expression can be used to determine the total number of Steiner topologies for n A-points, $F(n)$. Hwang *et al.* [19] give the following expression for the total number of topologies

$$F(n) = \sum_{k=0}^{n-2} \sum_{n_3=0}^{\lfloor (n-k-2)/2 \rfloor} \left(\begin{array}{c} n \\ n_3 \end{array} \right) F(n-n_3, k+n_3) \frac{(k+n_3)!}{k!}.$$

In this expression n_3 is the number of A-points that have degree three. The summations are over all possible number of S-points and possible number of A-points that can have degree three. The expression $F(n-n_3, k+n_3)$ gives

Number of A-points n	Number of full Steiner topologies $f(n)$
2	1
3	1
4	3
5	15
6	105
7	945
8	10,395
9	135,135
10	2,027,025
11	34,459,425
12	654,729,075
20	10^{20}
30	10^{37}
40	10^{55}
50	10^{74}

Table 2.1. The number of full Steiner topologies for different numbers of A-points.

the number of Steiner topologies where the n_3 A-points having degree three are treated as S-points. The function $\lfloor x \rfloor$ gives the largest integer less than or equal to x. Table 2.2 shows the total number of Steiner topologies for different numbers of A-points.

A relatively minimal tree with a given Steiner topology is called a *Steiner tree* or *ST*. All edges of the tree must have a non-zero length otherwise the tree is that for a different topology and not necessarily a Steiner topology. Zero length edges may occur when the position of an S-point coincides with the position of an A-point or another S-point. Figure 2.8 shows an example of two relatively minimal trees for different topologies. The topology on the left is a Steiner topology. As the relative positions of the points change the S-points S and T come closer together. When they coincide the length of the edge between the S-points is zero and the topology is no longer a Steiner topology and the relatively minimal tree is not a Steiner tree. When the topology is a full Steiner topology the relatively minimal tree is called a *full Steiner tree* or *FST*. The tree on the left in Figure 2.8 is a FST.

Winter proves that at most one ST exists for a given Steiner topology.[14] The proof is based on showing that a relatively minimal tree for a given topology is unique. This result and knowing that an SMT is an ST suggests that a way of finding a SMT is to construct all STs and choose the ST with the smallest length. Unfortunately it was shown above that the number of Steiner topologies is exceedingly large for all but very small problems (see Table 2.2).

[14]Lemma 10-1 Winter [42].

Number of A-points n	Number of Steiner topologies $F(n)$
3	4
4	31
5	360
6	5,625
7	110,880
8	2,643,795
9	74,035,080
10	2,382,538,725
20	10^{26}
30	10^{46}
40	10^{68}
50	10^{90}

Table 2.2. The total number of Steiner topologies for different numbers of A-points.

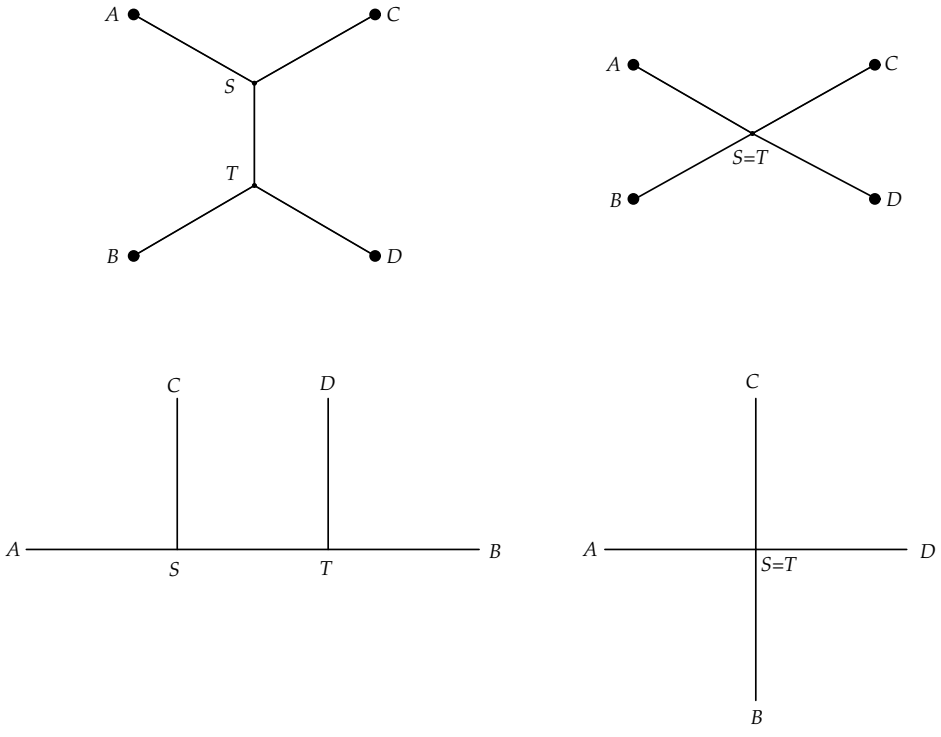


Figure 2.8. Two relatively minimal trees for different topologies. The positions of A, B, C and D have been changed until S and T coincide. The tree on the right is a relatively minimal tree for its topology shown below but the topology is not a Steiner topology and the tree is not a Steiner tree.

2.7 Decomposition of Steiner Trees

An important property of a Steiner topology is that it can be decomposed into one or more full Steiner topologies. For each full topology the FST can be constructed separately and all FSTs can be joined to give the ST for the original topology. An example of the decomposition of a Steiner topology is shown in Figure 2.9. Each ellipse contains n A-points (A through L) and $n-2$ S-points (U through Y). In this example the position of S-point U is determined by A-points B, C and D, and the positions of S-points V and W are determined by the positions of A-points D, E, F and L.

The decomposition theorem is:

> Let T be the SMT for the set of n points $\mathcal{A}$. For some *division* $(A_1, \ldots, A_t)$ of $\mathcal{A}$ where $\bigcup_{i=1}^{t} A_i = \mathcal{A}$, $|A_i| \geq 2 \; \forall \; i$ and $1 \leq t \leq n-1$, the SMT T is the union of FSTs T_i $i = 1, \ldots, t$, where each T_i is the SMT for A_i. A_i is called a *component* and the set $(|A_1|, \ldots, |A_t|)$ is called the *partition* of the division $(A_1, \ldots, A_t)$.[15]

This theorem provides an alternative method for constructing a SMT. Instead of finding all STs for all Steiner topologies, construct all FSTs for all full Steiner topologies of *every* subset of $\mathcal{A}$. For each subset A_i the shortest FST is T_i. The SMT of $\mathcal{A}$ is the shortest tree formed by feasible unions of the T_i. Although investigating all *full Steiner topologies* is less work than considering all *Steiner topologies* it is still an extremely large task. The number of full Steiner topologies grows rapidly with increasing problem size (see Table 2.1).

2.8 The Steiner Polygon

An important criterion for decomposing a Euclidean Steiner tree problem into smaller problems is the *Steiner polygon criterion*. This criterion is very powerful because the super-polynomial nature of the Euclidean Steiner tree problem means a huge reduction in computation time can be achieved by decomposition into smaller problems.

The SMT of $\mathcal{A}$ lies within the *convex hull* of $\mathcal{A}$, however a generally smaller region can be found which contains the SMT. Theorem 1.5 of Hwang *et al.* [19] uses the wedge and lune properties of a SMT (described in Section 2.5) and provides a method for generating the smaller region.

A convex hull of a set of points is the smallest convex set containing the set of points. The first plot in Figure 2.10 shows the convex hull of a set of ten points. The smallest hull is called the *Steiner hull* and the A-points on the boundary form the *Steiner polygon* or *SP*. A decomposition occurs when the SP of $\mathcal{A}$ is *degenerate*. When one or more A-points appear in the SP more than once then the SP is degenerate. The SP also provides valuable information about the maximum degree of A-points, and is used to eliminate from consideration divisions that give infeasible unions of component FSTs. The maximum degree

[15] Theorem 11-1 Winter [42].

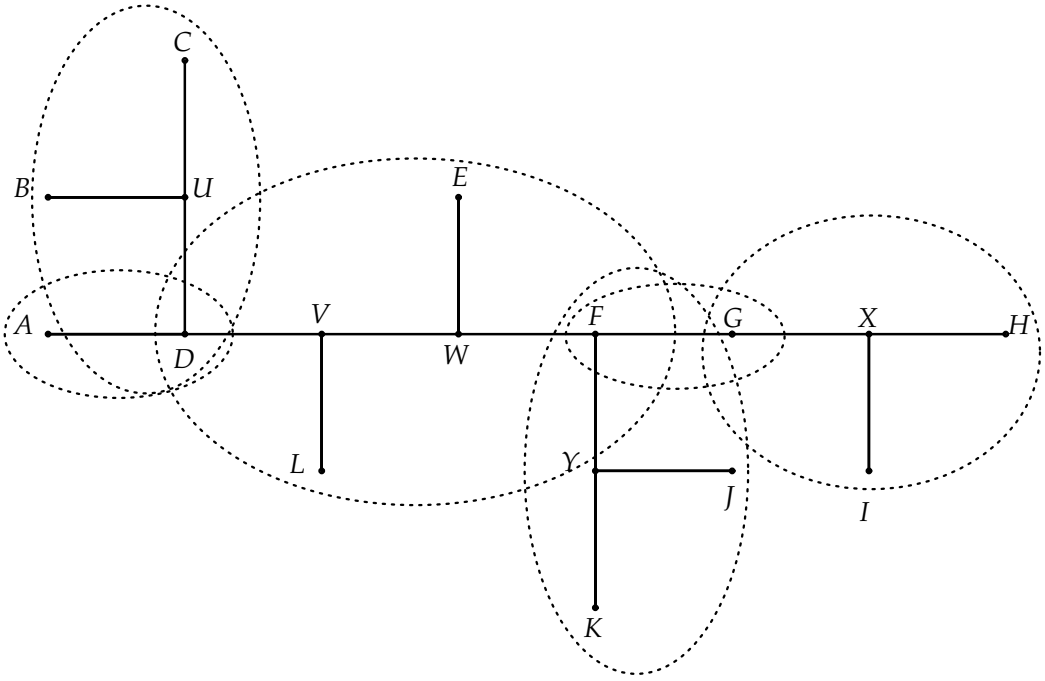


Figure 2.9. An example of the decomposition of a Steiner topology. The six regions each enclose a *full* Steiner topology. The largest is a four topology for connecting A-points D, E, F and L using two S-points V and W.

of each A-point in the SMT is found using the non-degenerate SP. If A-point a_i is *not* on the SP then the maximum degree of a_i is three. If a_i is on the SP and the interior angle of the SP at a_i is less than 120° then the maximum degree is one otherwise it is two.

2.8.1 Constructing the Steiner polygon

The SP is constructed iteratively beginning with the convex hull of $\mathcal{A}$, H_0. The following step is used to modify the current hull:

> If two adjacent A-points a_i and a_j on H_i and another A-point a_k (possibly already on H_i) are such that the triangle $a_i a_k a_j$ contains no other A-points and the interior angle $a_i a_k a_j$ of the triangle is greater than or equal to 120° then the edge $a_i a_j$ is replaced by edges $a_i a_k$ and $a_k a_j$ to give a new hull H_{i+1}. When no changes are possible the process stops.

The final hull obtained by the above process is the Steiner hull. This process is called *removing wedges*. A process to improve upon a SP found by removing wedges is called *removing quadrilaterals*. This is described in section 2.8.3 below.

Figure 2.10 shows the construction of a SP for a set of ten points. The first plot shows the convex hull of the set of ten points, $\{A, G, H, B, F, D\}$. The construction of the SP arbitrarily begins at A. The point adjacent to A (in a clockwise sense) is G. Point I is in a position such that the angle AIG is greater

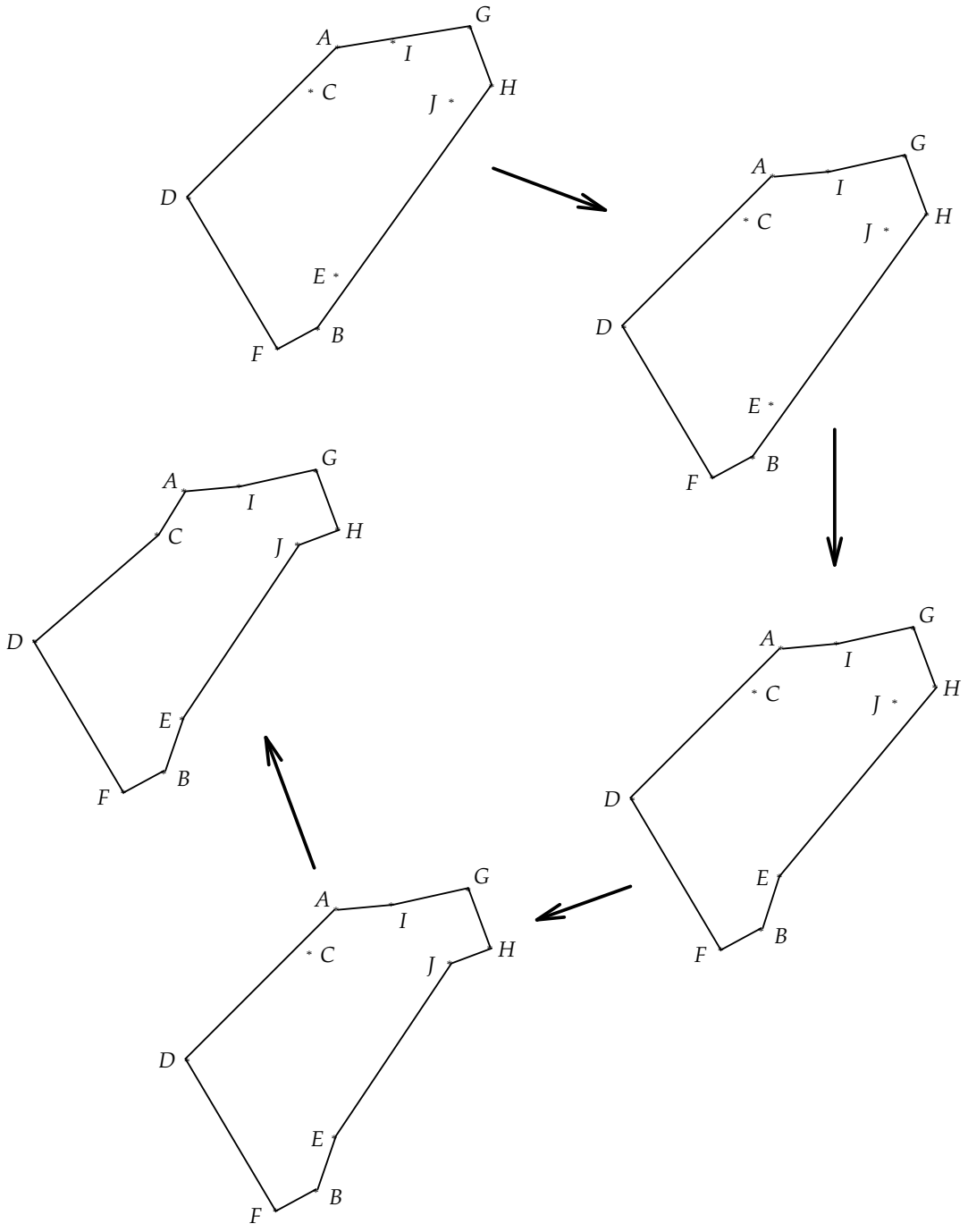


Figure 2.10. An example of the iterative construction of a SP. In this example all the A-points are on the Steiner polygon.

than 120° and the triangle AIG contains no other A-points. Therefore I is added to the hull to give $\{A, I, G, H, B, F, D\}$ (see the second plot in Figure 2.10). The process is repeated but now considering A and its new adjacent point I. There are no points satisfying the angle and empty triangle conditions. Attention now moves to I and G. Again no points can be found to add to the hull. The testing of conditions continues without success until H is reached. H and adjacent point B provide two candidates for inclusion E and J, both are such that the angle formed with H and B is greater than 120°, and neither triangle formed with H and B contains any other points. Either point can be added and E is arbitrarily chosen (see the third plot). In the next step J is added between H and E (fourth plot).[16] The last point to be added is C, between D and A (fifth plot). No more changes are possible and the fifth plot shows the SP $\{A, I, G, H, J, E, B, F, D, C\}$. In this particular example all ten points are on the SP. This is a fortunate result in terms of finding the SMT as it greatly reduces the number of possible topologies to be considered.

A SP is degenerate if it is self intersecting, that is one or more points appears more than once in the SP. Figure 2.11 shows the construction of a seven point degenerate SP. The convex hull contains only three points but the SP $\{A, C, F, C, G, B, G, E, D, E\}$ contains all seven points and is degenerate at three points (C, E and G). This decomposes the problem from a seven point problem to four smaller problems, three being trivial two point problems and the fourth a four point problem. The first degeneracy occurs when considering F and its adjacent point B. The point C which is already on the hull can be inserted between F and B (fifth plot of Figure 2.11). The seventh and eighth plots show the creation of the degeneracies at E and G respectively.

2.8.2 Some findings on Steiner polygons

This section presents some results on degeneracy and the proportion of points on SPs. This was performed to verify the same results described by Winter.[17] A listing of the program used to find Steiner polygons is in Appendix A. One hundred SPs for each of $n = 4, \ldots, 100$ were calculated where n is the number points. The points were randomly distributed in either the unit square or the unit circle.[18] For each distribution of points and value of n the probability of degeneracy and the average proportion of points on *non-degenerate* SPs were estimated. Figures 2.12 and 2.13 show the smoothed estimates of the probability of degeneracy and the average proportion of points on the polygon for non-degenerate SPs respectively. The smoothed estimates were obtained using a simple central moving average.

Winter only gives results for the unit circle distribution and then only for problems with 4 to 50 points. The above results agree with those of Winter.[19]

[16] The order of adding points is not important, choosing J first then E would give the same result. *The Steiner polygon of a set of points is unique.*

[17] Section 12.3 Winter [42].

[18] Both distributions are used because only after obtaining the unit square results was it realised that Winter had used the unit circle distribution for his SP experiments.

[19] Figures 12-4 and 12-5 Winter [42].

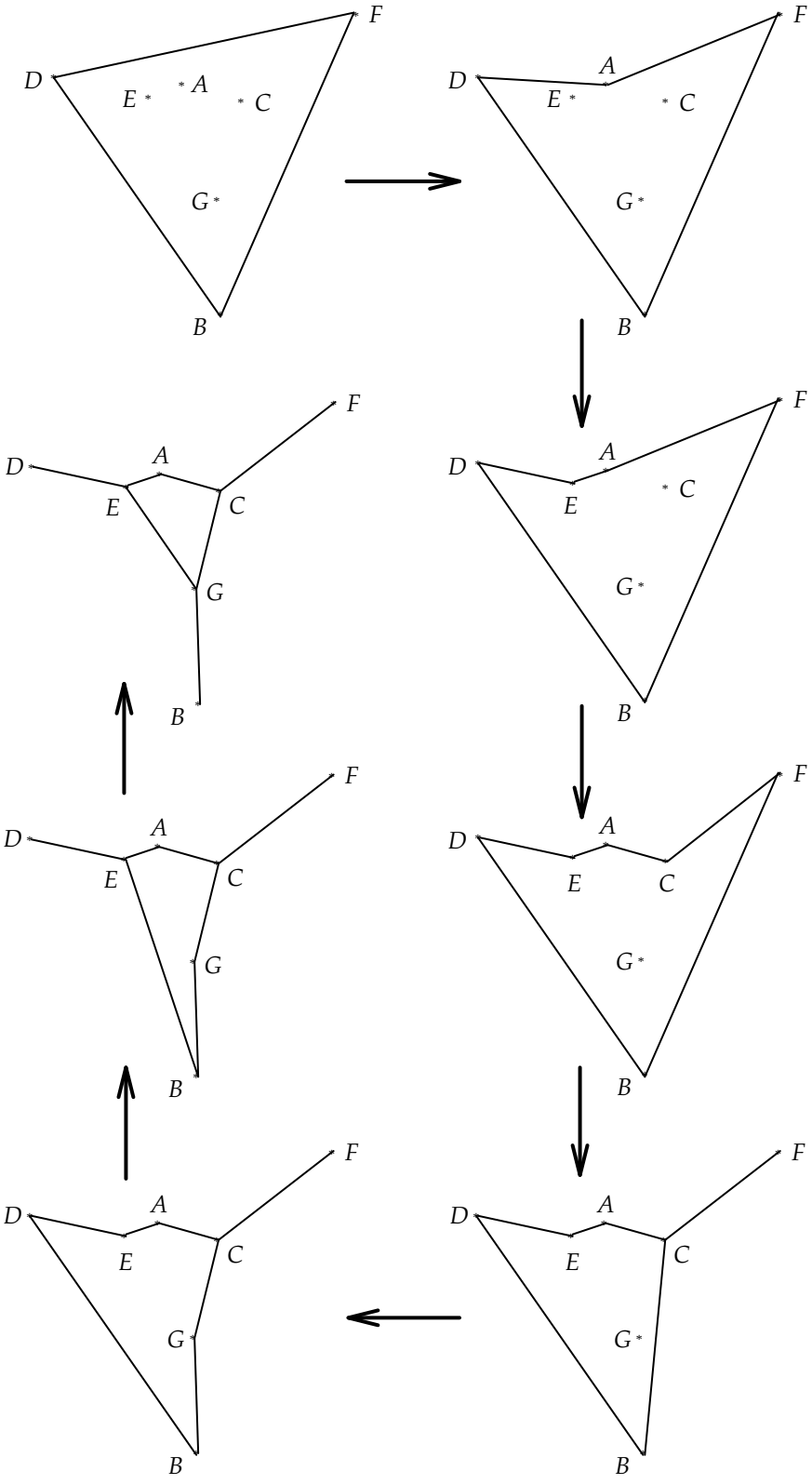


Figure 2.11. An example of the iterative construction of a degenerate SP. The polygon is degenerate at C, E and G.

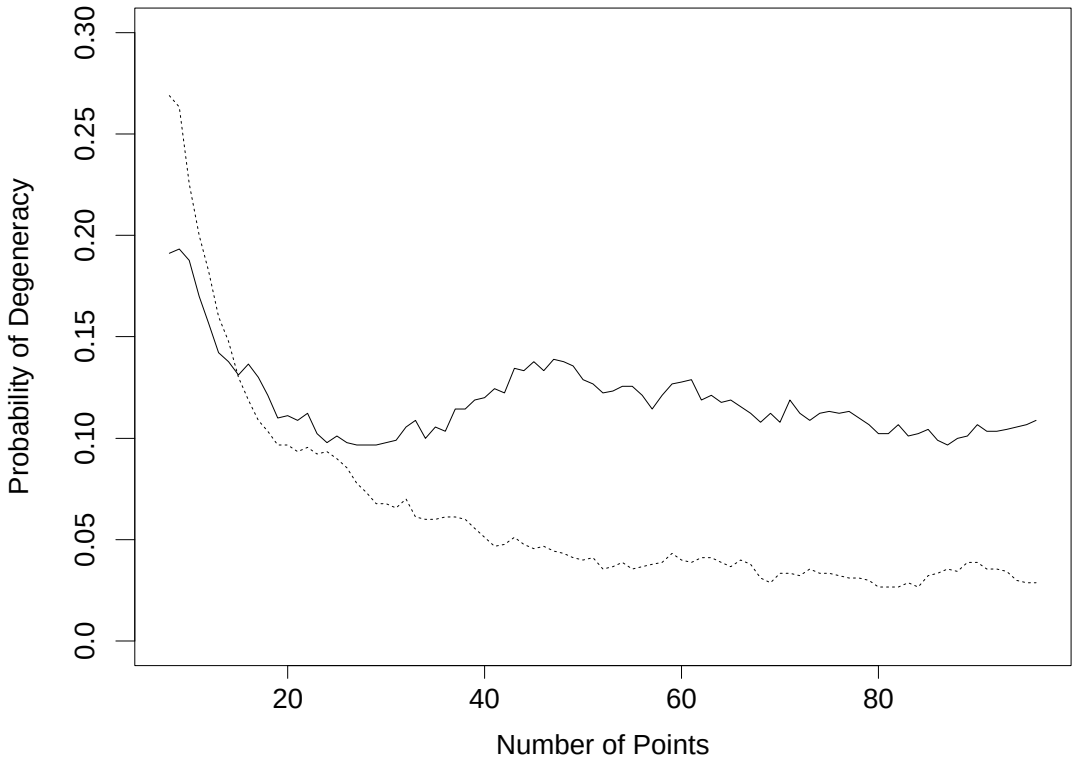


Figure 2.12. The probability of a degenerate SP for randomly selected sets of points. The solid line is for points distributed in the unit square, the dashed line for the unit circle.

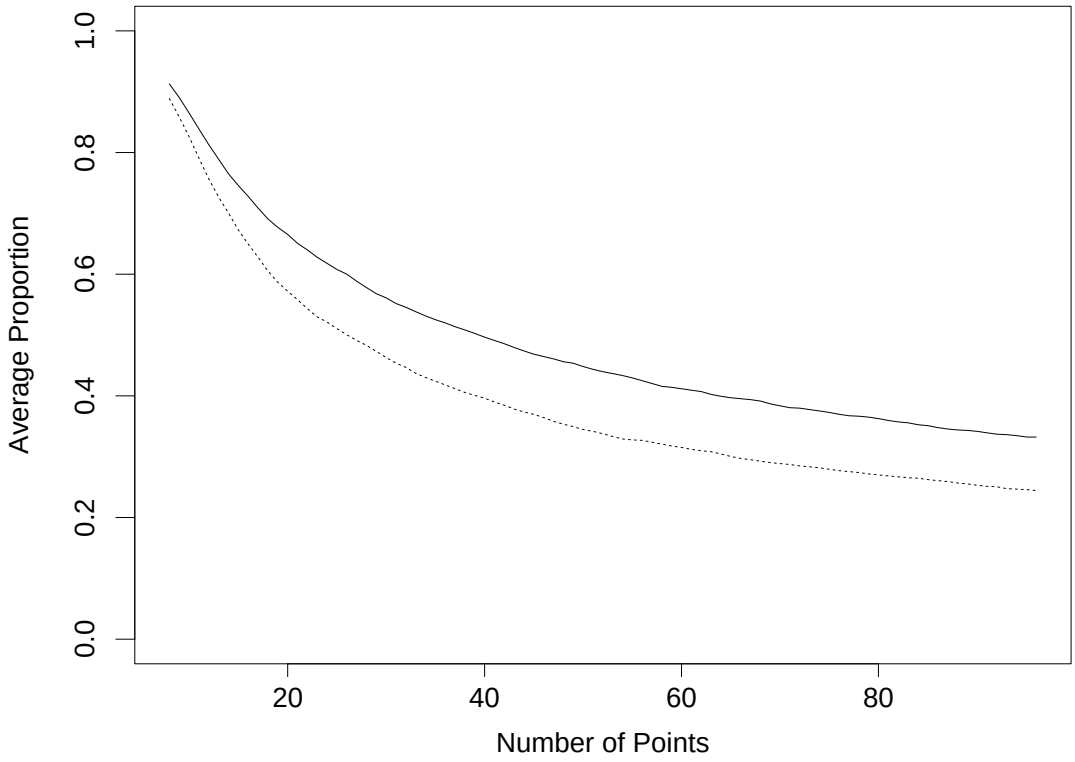


Figure 2.13. The average proportion of points on a non-degenerate SP for randomly selected sets of points. The solid line is for points distributed in the unit square, the dashed line for the unit circle.

The following comments are made:

- The probability of degeneracy for unit square SPs is higher than for unit circle SPs and appears constant for high n. Whereas the unit circle probability tends towards zero or a small constant value as n increases;
- The average proportion of points on a non-degenerate SP for points distribute randomly in the unit square is higher for all values of n than the average proportion for unit circle non-degenerate SPs.

Little effort is expended in attempting to explain these differences. Obviously it is to do with the differing shape and size of the regions and in particular the fact that a square has corners and a circle does not.[20]

2.8.3 Removing quadrilaterals

Theorem 1.6 of Hwang *et al.* [19] provides a method for removing a quadrilateral to give a smaller SP. It is more complicated than removing wedges but when successful gives a degenerate SP and provides a valuable decomposition of a problem. The theorem is:

> If four points a, b, c and d are four points on the Steiner hull satisfying the following conditions:
>
> - a and b are A-points;
> - $abcd$ is a convex quadrilateral with interior angles dab and abc both greater than or equal to 120°;
> - quadrilateral $abcd$ contains no other A-point;
> - the angle $bxa \geq dab + abc - 150°$ where x is the intersection of the diagonals of the quadrilateral.
>
> Then the quadrilateral $abcd$ can be removed from the hull, and no part of the SMT can lie inside the quadrilateral.

The theorem requires that only a and b are A-points, c and d can be any points on the hull, that is A-points or points on edges joining A-points. Figure 2.14 shows the removal of a quadrilateral. In this example the four points satisfy the conditions of the theorem. The region R originally enclosed by the SP (found by removing wedges) is divided into two regions R_1 and R_2 and the line joining a and b. Therefore the SMT problem is decomposed into two smaller problems: finding the SMT of A-points belonging to R_1, and similarly for R_2. The SMT of the original problem is the two smaller SMTs connect by the segment ab. The removal of wedges does not give a decomposition (assuming c and d are A-points) because neither angle dac nor angle dbc are greater than or equal to 120°.

An extension of the theorem suggested by Sarkar [33] is to allow a and b to be *any* A-points, instead of A-points *on the SP*. A brief investigation failed to

[20] This short section was merely to verify Winter's results, not to try to explain the differences between squares and circles.

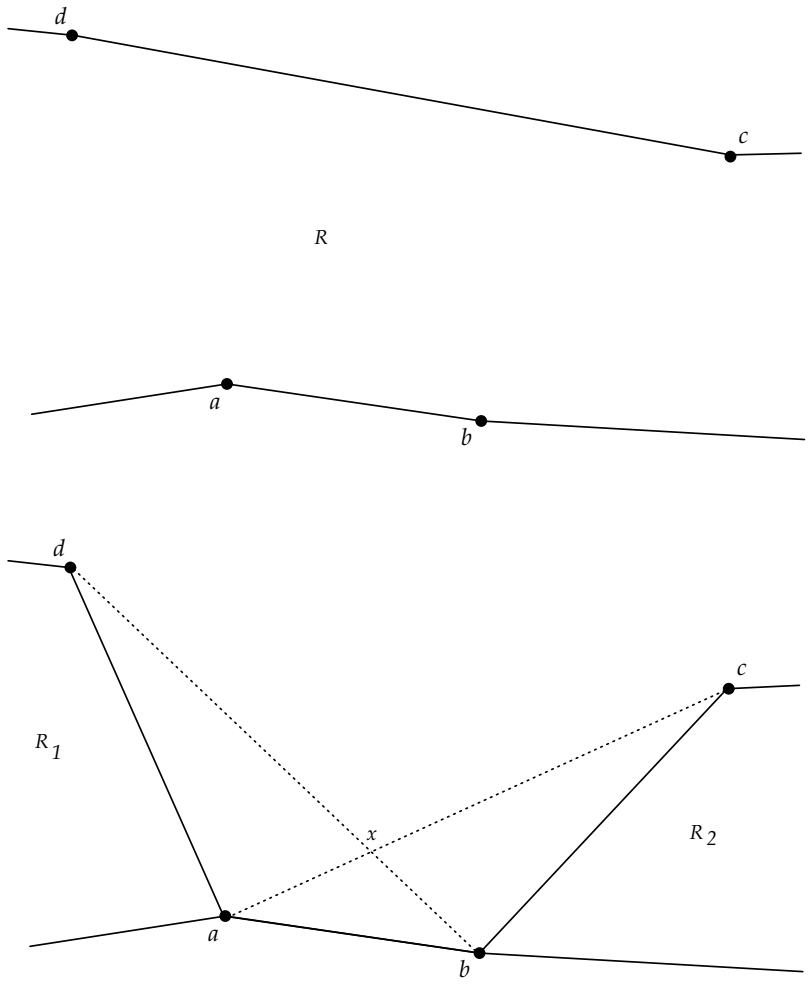


Figure 2.14. Removing a quadrilateral from a SP.

find a counter-example. And a problem was created that had a SMT within the SP where the SP was found using the modified theorem. Hwang [21] states that no proof that he knows of exists for the modified theorem. Further study is required to gain insight into this aspect of the Steiner polygon.[21]

2.9 Construction of Full Steiner Trees

The construction of FSTs is described in three sections below. The first looks at the simple three point problem, the second at the four point FST and the third at the linear time construction algorithm of Hwang [20] and hereafter called *Hwang's linear time FST algorithm* which attempts to find the FST for five or more points. But before studying any of these construction methods definitions of basic elements used in FST construction are required.

2.9.1 Further definitions

If two A-points a_i and a_j are joined to S-point s then the edges a_is and a_js must make an angle of 120° with each other at s. The point s is either on the arc a_ia_j of the circumscribing circle of the equilateral triangle defined by $a_ia_je_{ij}$ where e_{ij} is to the left of the segment a_ia_j and s is on the right, *or* on the arc a_ja_i of the circumscribing circle of the equilateral triangle defined by $a_ia_je_{ji}$ where e_{ji} is to the right of the segment a_ia_j and s is on the left. The points e_{ij} and e_{ji} are called *equilateral points* or *E-points*.

E-points are the basic building blocks of FST construction. A *simple E-point* is the third point of the equilateral triangle formed by two A-points. Each pair of A-points a_i, a_j define two E-points. The E-point to the left of the line segment a_ia_j looking from a_i towards a_j is denoted by (a_i, a_j), the point to the right is (a_j, a_i). The E-points are said to be *based* on a_i and a_j.

The circles circumscribing the equilateral triangles $a_ia_j(a_i, a_j)$ and $a_ia_j(a_j, a_i)$ are called *equilateral circles* or *E-circles*. The 120° arcs a_ia_j and a_ja_i are called *equilateral arcs* or *E-arcs*. These various objects are shown in Figure 2.15.

Compound E-points or *higher order E-points* are based on one A-point and one E-point, or two E-points. The A-points making up an E-point are called the E-point's *terminal points*. Figure 2.16 shows the construction of two compound E-points: $e_1 = ((a_4, a_3), a_2)$ has terminal points a_2, a_3 and a_4, and $e_2 = (e_1, (a_1, a_5))$ has terminal points $a_1, \ldots, a_5$.

2.9.2 FST for three points

For three points a_1, a_2 and a_3 there is only one full Steiner topology to be investigated. If the unique FST exists then its only S-point s must lie on either the E-arc a_1a_2 or E-arc a_2a_1 and the edges a_1s, a_2s and a_3s must make a 120° angle with each other. If the FST does not exist then the SMT is the MST of a_1,

[21] A complete description of the quadrilateral decomposition can be found in Hwang *et al.* [18].

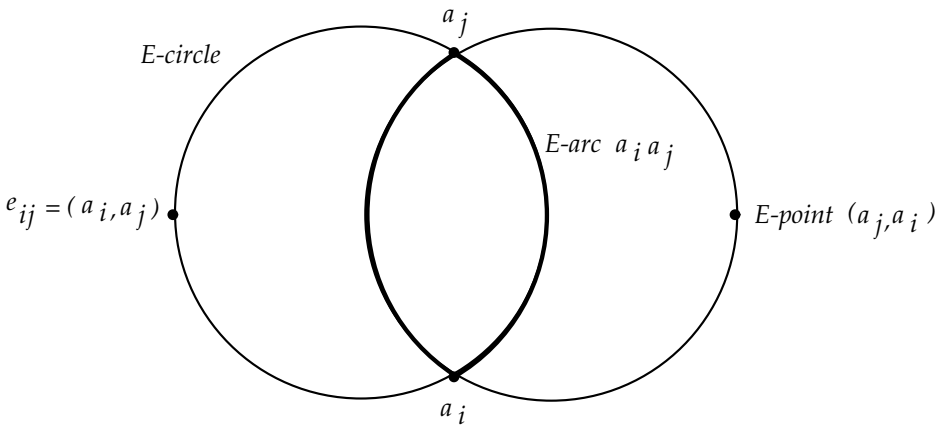


Figure 2.15. An illustration of the various geometrical objects defined by two A-points.

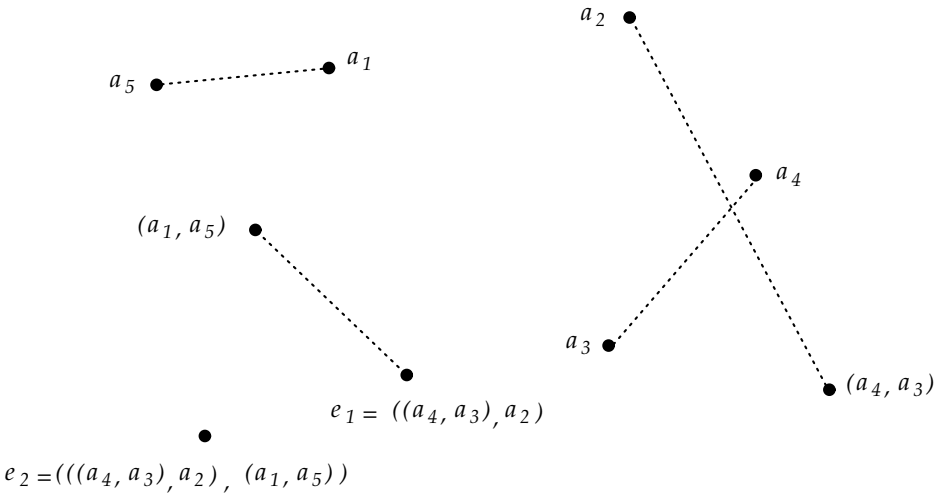


Figure 2.16. An example of compound E-points. E-point e_1 is based on A-point a_2 and E-point (a_4, a_3). The E-point is on the left when looking from (a_4, a_3) towards a_2.

a_2, a_3. Figures 2.17 and 2.18 show the construction of a three point FST and the attempt to construct a non-existent FST respectively. In Figure 2.17 the segment $(a_2, a_1)a_3$ is called the *axis* of the FST. The length of the axis is equal to the length of the FST. The axis is one of the three Simpson lines (see Section 2.1).

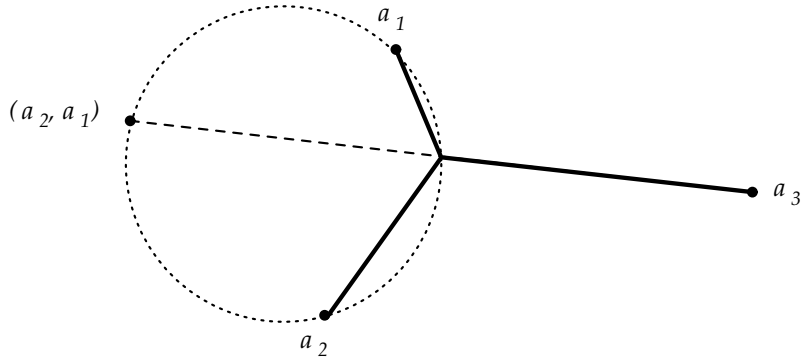


Figure 2.17. An example of construction of a three point FST.

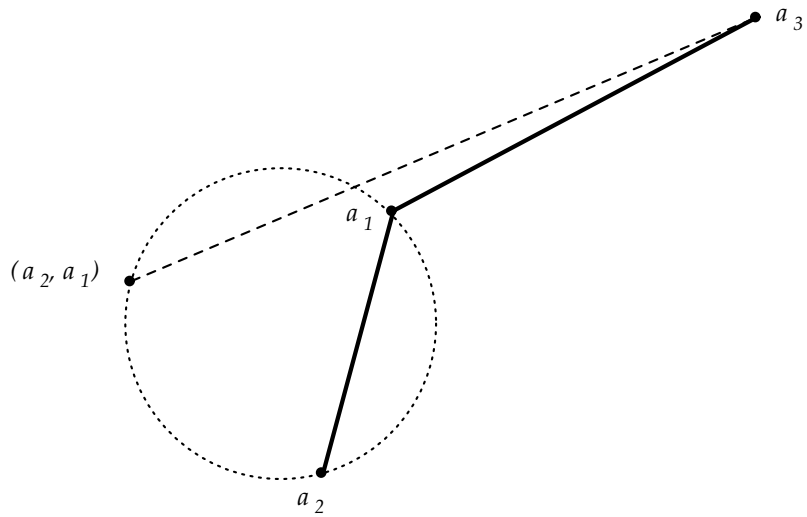


Figure 2.18. An example of the non-existence of a three point FST. The axis $(a_2, a_1)a_3$ does not intersect the E-arc.

The FST of Figure 2.17 can be constructed in two other ways. Instead of using E-point (a_2, a_1), E-points (a_3, a_2) or (a_1, a_3) can be used to give axis $(a_3, a_2)a_1$ or $(a_1, a_3)a_2$ respectively. The three possible constructions are equivalent. The three axes are equivalent because the cyclic orders of the A-points are identical, $213 \equiv 321 \equiv 132$. Importantly, this cyclic ordering corresponds to the clockwise order of the A-points induced by the Steiner polygon.

There are two possible three point full Steiner topologies (see Figure 2.19). The clockwise cyclic orderings of the A-points are different. The first topology gives orderings $123 \equiv 231 \equiv 312$, the second $132 \equiv 321 \equiv 213$. The orderings are used to form E-points. The *bracketing* of two A-points, an A-point and

E-point or two E-points generates an *association*. An association describes an axis. The *equivalent associations* generated by the two topologies are:

$$(a_1, a_2), a_3 \equiv (a_2, a_3), a_1 \equiv (a_3, a_1), a_2 \equiv a_1, (a_2, a_3) \equiv a_2, (a_3, a_1) \equiv a_3, (a_1, a_2)$$
$$(a_1, a_3), a_2 \equiv (a_3, a_2), a_1 \equiv (a_2, a_1), a_3 \equiv a_1, (a_3, a_2) \equiv a_3, (a_2, a_1) \equiv a_2, (a_1, a_3)$$

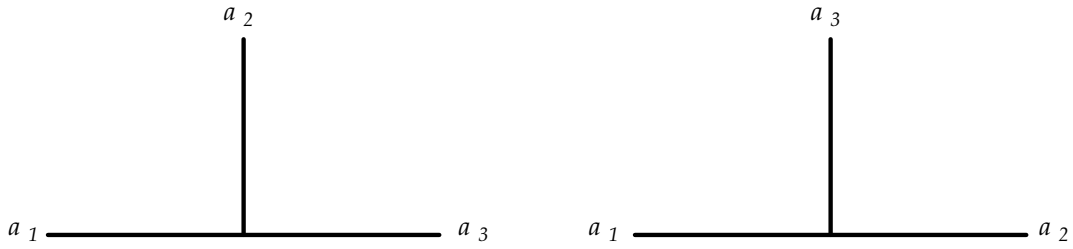


Figure 2.19. The two possible three point full Steiner topologies. The clockwise cyclic orderings are different: 123 and 132.

To find the FST of three points, if it exists, an attempt is made to construct the FST of *one* of the full topologies using *one* of the equivalent associations. The topology used depends on the cyclic ordering induced by the Steiner polygon.

2.9.3 FST for four points

The construction of a four point FST is only slightly more complicated than the three point FST construction. The number of four point full Steiner topologies is three (see Figure 2.20). Therefore there are three sets of equivalent associations, for example the first topology of Figure 2.20 generates the following twenty equivalent associations:

$$\begin{aligned}
&(a_1, a_2), (a_3, a_4) \equiv (a_3, a_4), (a_1, a_2) \equiv (a_1, (a_2, a_3)), a_4 \equiv a_4, (a_1, (a_2, a_3)) \\
\equiv\ &((a_1, a_2), a_3), a_4 \equiv a_4, ((a_1, a_2), a_3) \equiv a_1, (a_2, (a_3, a_4)) \equiv (a_2, (a_3, a_4)), a_1 \\
\equiv\ &a_1, ((a_2, a_3), a_4) \equiv ((a_2, a_3), a_4), a_1 \equiv (a_2, a_3), (a_4, a_1) \equiv (a_4, a_1), (a_2, a_3) \\
\equiv\ &((a_4, a_1), a_2), a_3 \equiv a_3, ((a_4, a_1), a_2) \equiv (a_4, (a_1, a_2)), a_3 \equiv a_3, (a_4, (a_1, a_2)) \\
\equiv\ &a_2, (a_3, (a_4, a_1)) \equiv (a_3, (a_4, a_1)), a_2 \equiv a_2, ((a_3, a_4), a_1) \equiv ((a_3, a_4), a_1), a_2
\end{aligned}$$

Fortunately constructing a four point FST can be viewed in a different way and associations can be pushed aside for the moment (Sarkar [33]). Consider a quadrilateral of the form shown in Figure 2.21 for points a, b, c and d and a topology specifying that a, b are connected to S-point u and c, d are connected to S-point v. Three conditions must be satisfied for the FST to exist, they are:

- The quadrilateral must be convex. The diagonals ad and bc must cross;
- The axis must intersect segments ab and cd. The axis of the FST is the segment from E-point (b, a) to E-point (c, d);
- The E-circles must not intersect.

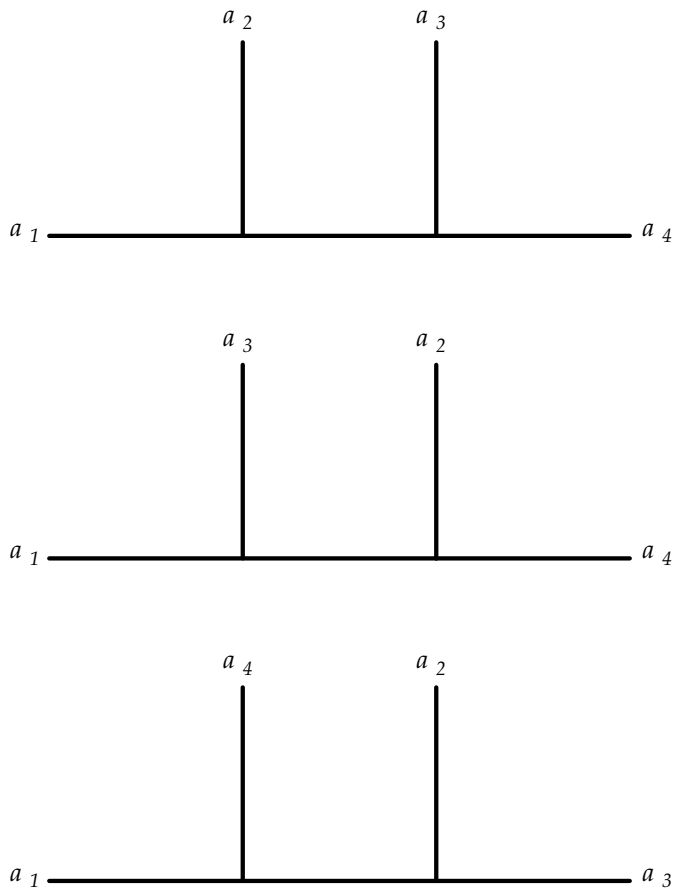


Figure 2.20. The three possible four point full Steiner topologies.

If all three conditions are satisfied then u and v are given by the intersection of the axis with the E-arcs (b, a) and (c, d) respectively. The FST of the points in Figure 2.21 with the given topology is shown in Figure 2.22.

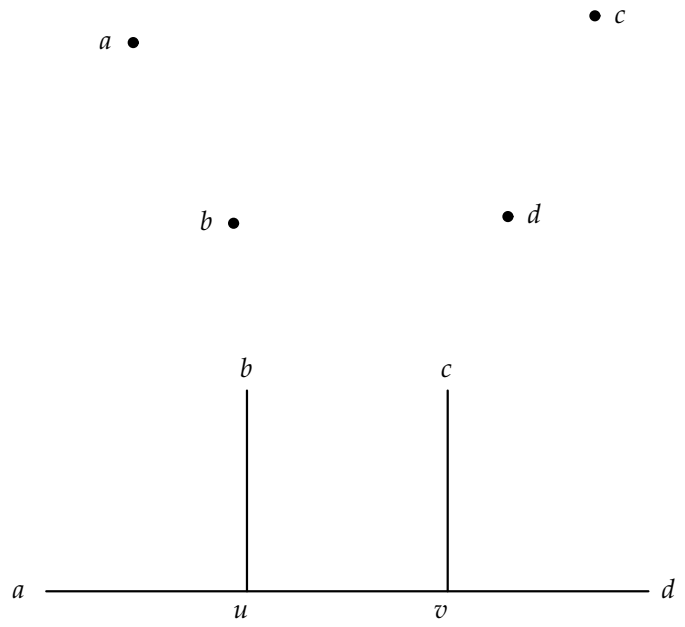


Figure 2.21. Four points and a particular Steiner topology.

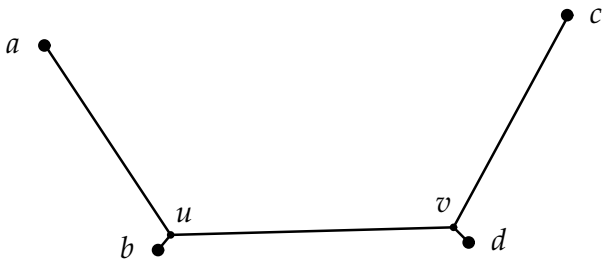


Figure 2.22. The four point FST corresponding to the points and topology in Figure 2.21.

The second condition is a simpler statement of the need for the axis to intersect the E-arcs. Violation of the third condition gives a network that is not a Steiner topology (see Figure 2.23).

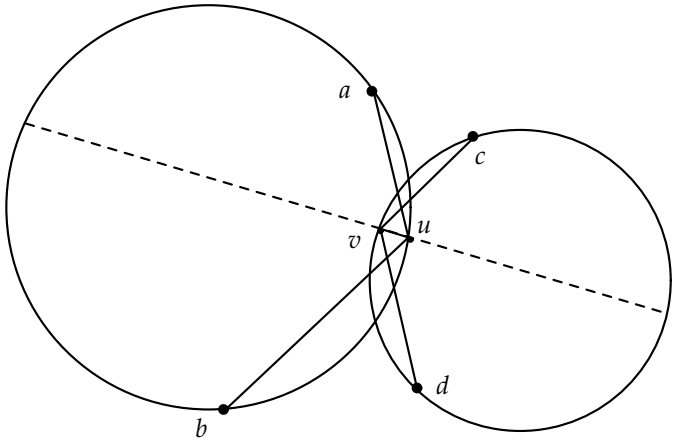


Figure 2.23. An example of the violation of the third condition.

2.9.4 Hwang's linear time FST algorithm

This section describes an algorithm to find the FST for a given topology with five or more A-points. The algorithm is due to Hwang [20] and has a $O(n)$ time requirement. The algorithm iteratively reduces a topology by replacing an S-point of unknown position by an E-point formed by the A-points or E-points (or both) connected to the S-point in the topology. The position of the E-point, that is if it is to the left or right, is based on the relative positions of points in the topology. If at any stage a topology can not be reduced then the FST does not exist. Once a topology has been reduced to four points (a combination of A-points and E-points) then the four point FST method described above can be used. If this FST is found then a process of "unreducing" or expanding E-points to give S-points is applied. If an S-point can not be found then the FST does not exist.

Reduction

Given a full Steiner topology T with $n \geq 5$ A-points. Choose an arbitrary A-point r. Find the A-point farthest from r in T and call this point a, where "distance" is measured by the number of edges in the unique path from one point to another in the topology. There will always be at least two choices for a, which is taken is of no consequence. Because $n \geq 5$ the distance from r to a is at least three and there is at least two S-points s_1 and s_2 on the path from r to a. The S-point adjacent to a is called s_1. Adjacent to s_1 is A-point b and the S-point s_2. The point adjacent to s_2 not on the path from r to a is labelled v. This point can either be an A-point or a S-point. If it is a S-point it has two adjacent A-points, c and d. Figure 2.24 shows the labelling of points of a topology. The first is a five point topology where v is an A-point. The second topology is a six point topology where v is a S-point.

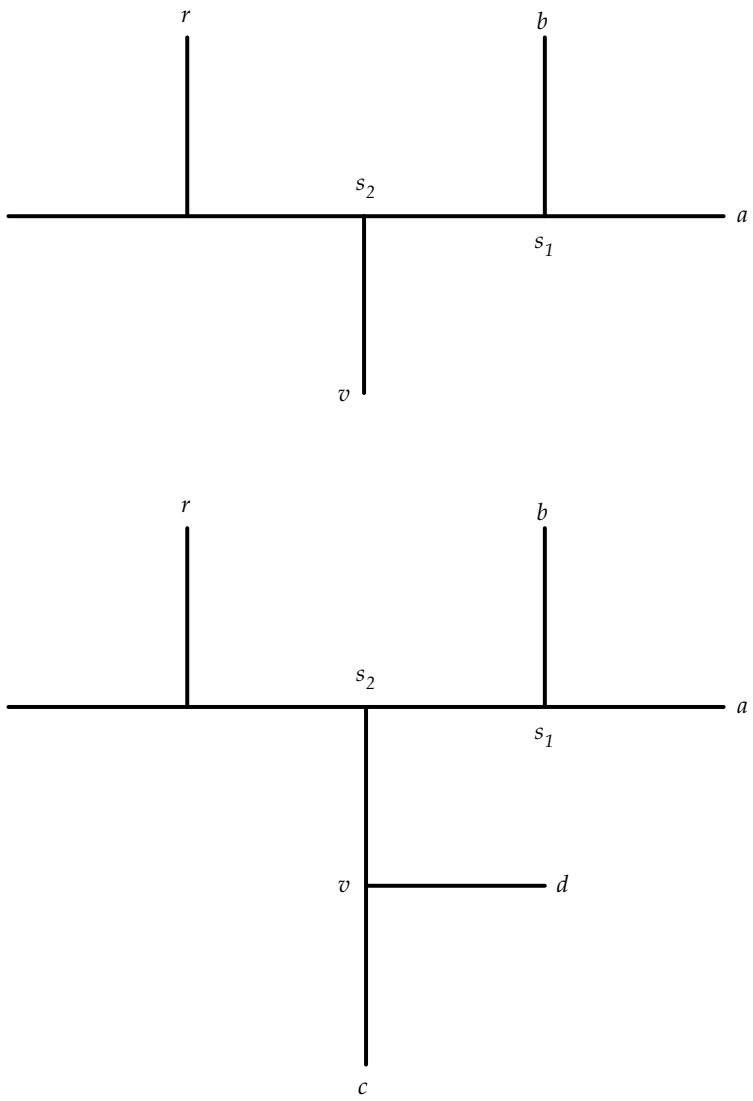


Figure 2.24. Examples of the labelling of topologies for Hwang's linear time FST algorithm. In the top topology c and d do not exist because v is not a S-point, compare this with the bottom topology.

The rules for reducing the topology are:

1. If v is an A-point:
   - Remove a, b and s_1 from T;
   - Add an E-point based on a and b and an edge connecting the E-point and s_2. The E-point is such that it is on the opposite side of the line ab from v.
2. If v is a S-point and c and d lie on the same side of the line ab:
   - Remove a, b and s_1 from T;
   - Add an E-point based on a and b and an edge connecting the E-point and s_2. The E-point is such that it is on the opposite side of the line ab from c.
3. If v is a S-point and c and d lie on opposite sides of the line ab:
   - Remove c, d and v from T;
   - Add an E-point based on c and d and an edge connecting the E-point and s_2. The E-point is such that it is on the opposite side of the line cd from a.
4. If none of the above three conditions apply then the lines ab and cd must cross and the FST does not exist.

The labelling and reduction is repeated using the same r until four points remain in T.[22] The FST of the four point topology is found, if it exists, using the four point method described above.

An example of a successful reduction is presented using the points and topology shown in Figure 2.25. The A-points are $1, \ldots, 6$ and the S-points are w, x, y and z. The positions of the S-points are not known. A-point 2 is arbitrarily selected to be r. The farthest points in the topology from 2 are 1 and 6, again arbitrarily, 1 is chosen to be a and 6 is b. The S-points z and y are s_1 and s_2 respectively. The point adjacent to s_2 not on the path from r to a is the A-point 5, labelled v. The labelled points and topology are shown in Figure 2.26. The points satisfy the first condition of the rules for determining the reduction: v is an A-point. The points 1, 6 and z are removed and replaced by an E-point that must be on the opposite side of the line ab from v. Therefore E-point $(1, 6)$ is constructed and connected to y in the topology. If the FST exists then the S-point z must lie on the E-arc $(1, 6)$.

The topology is now a five point full Steiner topology. The labelling and reduction is repeated. With point 2 as r the farthest points are 3, 4, 5 and $(1, 6)$. Arbitrarily $(1, 6)$ is chosen as a and therefore 5 is b. The S-points y and x are s_1 and s_2 respectively. The point adjacent to s_2 not on the path from r to a is S-point w, this is v. The labelled topology is shown in Figure 2.27. The point v is a S-point and c and d lie on the same side of the line ab. Therefore $(1, 6)$,

[22] The four points will be at made up of at least one A-point, r, and E-points.

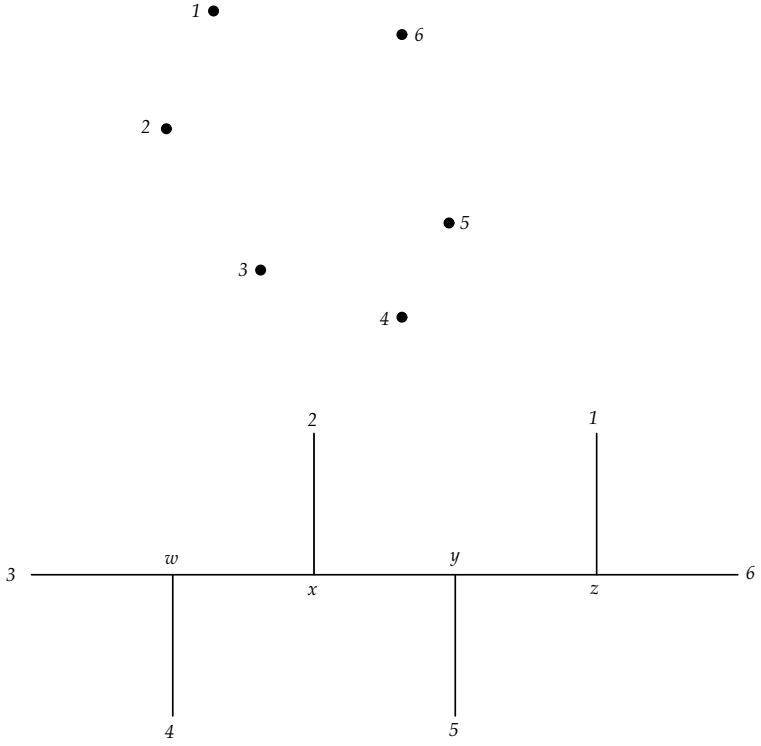


Figure 2.25. Topology reduction: the original six point topology and positions of points in the Euclidean plane.

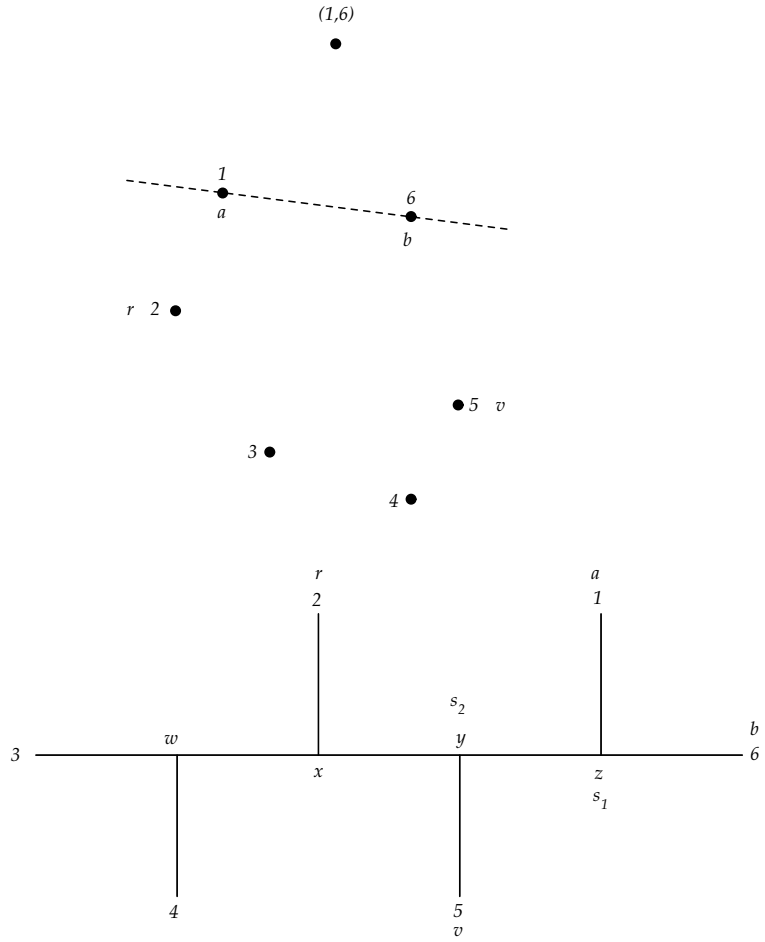


Figure 2.26. Topology reduction: the labelling of points and replacement of S-point z by E-point $(1, 6)$.

5 and y are removed from the topology and replaced by an E-point that is on the opposite side of the line ab from c, this is E-point $((1,6),5)$. For the FST to exist the S-point y must lie on the E-arc $((1,6),5)$. The reduced topology is shown in Figure 2.28. The reduction process ceases because the topology has been reduced to a four point topology with points 2, 3, 4 and $((1,6),5)$.

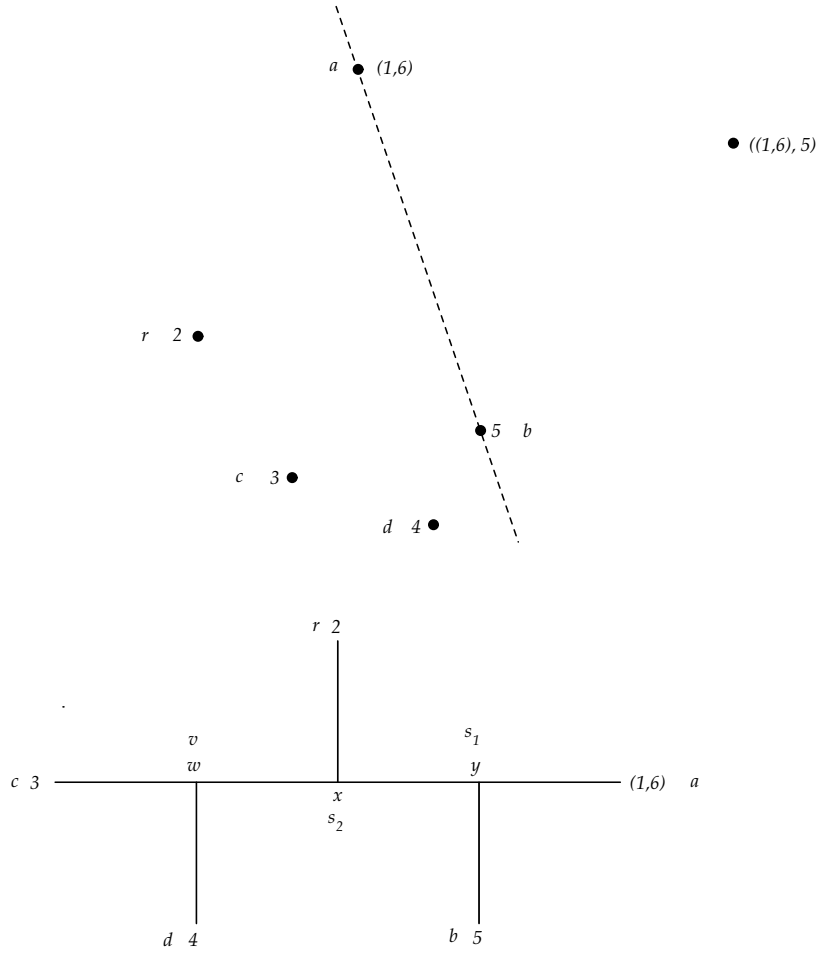


Figure 2.27. Topology reduction: the labelled reduced five point topology and replacement of S-point y by E-point $((1,6),5)$

The four point FST method described above finds the FST for the given topology. The FST is shown in Figure 2.29. Not shown in the figure is the axis of the FST. The axis is the segment from E-point $(2,((1,6),5)))$ to E-point $(4,3)$. The length of the segment is the length of the FST of the original six point full Steiner topology. Although at this stage it is not known if the FST exists. This is dependent on the successful generation of the S-points y and z.

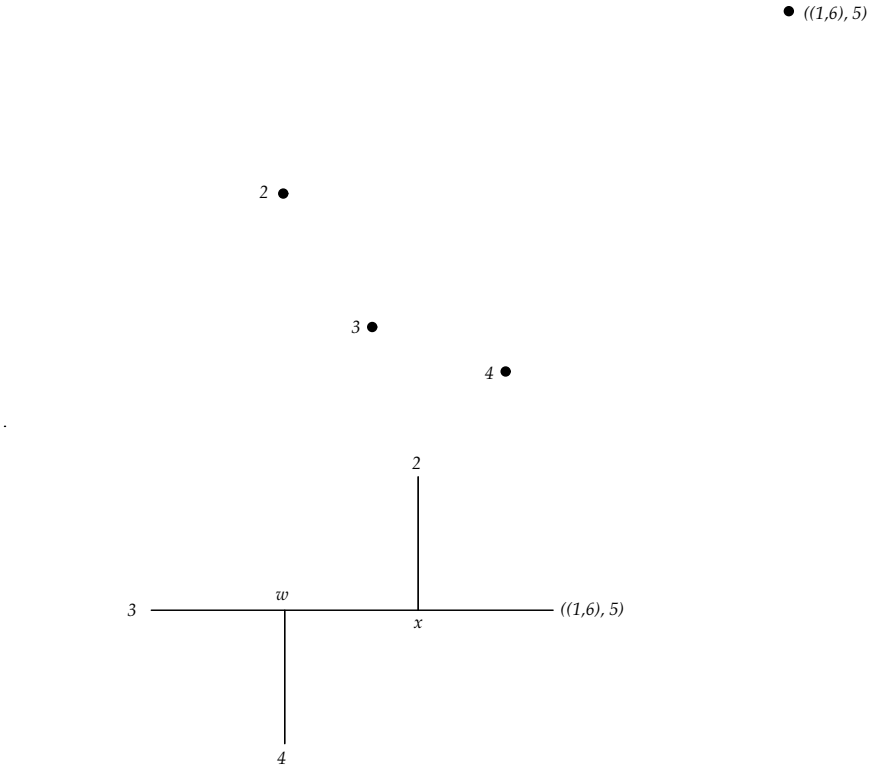


Figure 2.28. Topology reduction: the reduced four point topology and points.

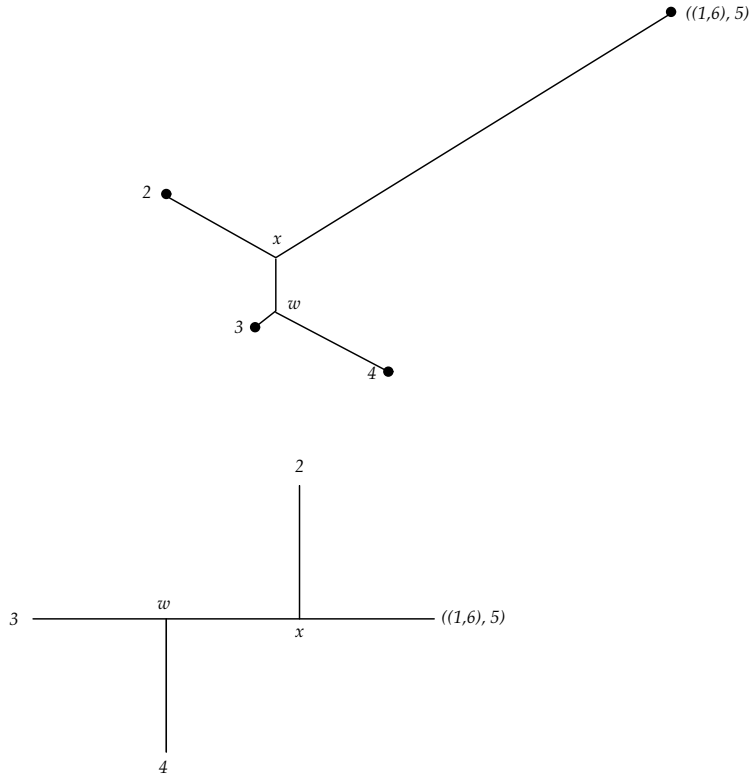


Figure 2.29. The four point topology FST.

Expansion

Given a successfully constructed four point FST it is necessary to backtrack and expand the E-points into S-points. This is relatively straightforward and is equivalent to finding the S-points of a succession of three point problems. If it is not possible to find a S-point then the FST for the original topology does not exist. The expansion is shown by example using the reduced topology of the previous section.

Figure 2.29 shows a segment from x to $((1,6),5)$. This is equivalent to the axis of a three point problem for points x, 5 and $(1,6)$. The S-point y must be the intersection of the axis and the E-arc $((1,6),5)$ if the the three point FST is to exist. Figure 2.30 shows the expansion of E-point $((1,6),5)$ to give S-point y.

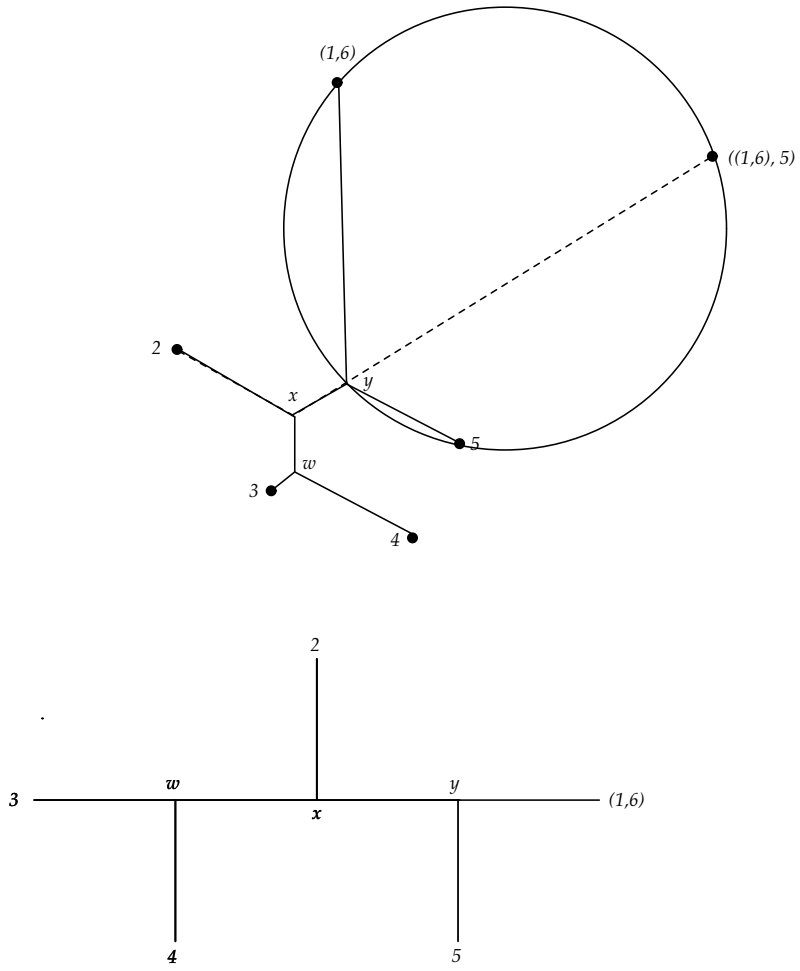


Figure 2.30. Topology expansion: E-point $((1,6),5)$ is replaced by S-point y.

The second and final expansion is to find S-point z by expanding E-point $(1,6)$. Point z is the intersection of the segment $y(1,6)$ and the E-arc $(1,6)$. Figure 2.31 shows this expansion and the FST of the original topology.

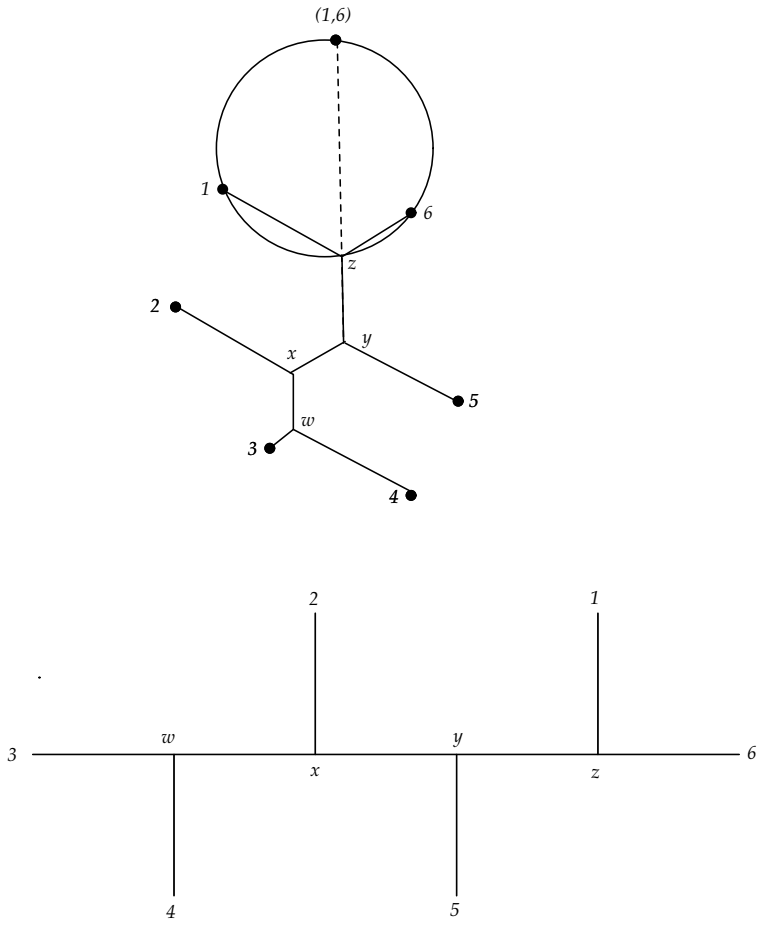


Figure 2.31. Topology expansion: E-point $(1,6)$ is replaced by S-point z and the six point FST is found.

2.9.5 Summary of FST construction methods

The construction of three, four and five or more point FSTs with a given full Steiner topology has been described above.[23] The three point FST method is very straightforward and involves two steps: finding an E-point then attempting to find the S-point. The four point FST is slightly more complicated, it involves finding two E-points, the axis and finally the S-points. There are several conditions for the existence of the four point FST with a given topology, namely the quadrilateral is convex, the axis intersects the segments joining the points giving the respective E-points, and the E-circles (and E-arcs) do not intersect. The method for constructing five or more point FSTs is the most complex method. It is necessary to reduce the topology to a four point full Steiner topology using rules based on the relative positions of points. The four point FST is found using the four point method and then the E-points formed during the reduction phase are expanded to give the S-points and the FST of the original full Steiner topology. The process of finding the FST stops when it is not possible to reduce the topology any further or an S-point can not be found, and therefore the original FST does not exist. Or the original FST is successfully constructed.

2.10 Optimal Algorithms

Several optimal algorithms have been proposed and implemented for the Euclidean Steiner tree problem over the years. All are cursed by the NP-hardness of the problem to be super-polynomial. However clever programming and geometry have combined to give implementations capable of solving up to 30–40 point problems in reasonable amounts of time and some 100 point problems if given enough time.[24] The first algorithm implemented is Cockayne's algorithm and is described in some detail. The amount of processing is evident from the description and the super-polynomial nature is apparent. The other main type of algorithm is GEOSTEINER. This algorithm and its successors are also described.

2.10.1 Cockayne's algorithm

The first algorithm for the Euclidean Steiner tree problem was proposed by Melzak. It was first implemented by Cockayne and has been improved upon by Cockayne other researchers.[25] The best implementation can solve up to twelve point problems in reasonable time. The algorithm attempts to construct FSTs for all possible subsets of the points and amalgamate the FSTs to form the SMT. At its core is the decomposition theorem described in Section 2.7. The algorithm is generally known as *Cockayne's algorithm*. In this section a description is given

[23]The construction of the two point FST has not been discussed. It is a trivial problem. The shortest connecting network for two points is a straight line from one point to the other.

[24]Reasonable being of the order of tens of minutes and enough being 12 hours.

[25]Section 2.1 Hwang *et al.* [19].

of the algorithm as described by Winter [42]. This algorithm is the improved version of Cockayne's algorithm called STEINER73.[26]

The decomposition theorem states that the SMT of a set of points is the union of FSTs of subsets. Therefore to find the SMT of a set of points $\mathcal{A}$ it is necessary to construct all divisions $(A_1, \ldots, A_t)$ of $\mathcal{A}$ and for each component A_i find its shortest FST T_i, called the minimum length FST. If all T_i of a division exist then the union $\bigcup_{i=1}^{t} T_i$ is a candidate for the SMT of $\mathcal{A}$.

To reduce the computational effort a range of tests exist for eliminating divisions from consideration. Some tests use properties of the Steiner polygon of $\mathcal{A}$ and A_i and the minimum spanning tree of $\mathcal{A}$, other tests are geometrical or combinatorial. The MST is the first candidate for the SMT, and the SP is used to decompose the problem, if fortunate, and provides information about the maximum degree of each point in the SMT (see Section 2.8).

The consideration of all divisions of $\mathcal{A}$ is streamlined by constructing divisions in an order dependent on the number of points in the components of a division. The *partition* of division $(A_1, \ldots, A_t)$ is $(|A_1|, \ldots, |A_t|)$, and fundamental relationships between the partition and number of edges in a tree made up of FSTs of the division provide rules for determining allowable partitions. Each component A_i has a minimum length FST or *MLFST* T_i, assuming it exists. The union of MLFSTs $T = \bigcup_{i=1}^{t} T_i$ is a tree connecting $\mathcal{A}$ if $\bigcup_{i=1}^{t} A_i = \mathcal{A}$ and $|A_i| \geq 2 \; \forall \; i$. Because T is a tree connecting n A-points and is the union of t FSTs containing a total of $\sum_{i=1}^{t}(|A_i| - 2)$ S-points, it must have $n + \sum_{i=1}^{t} |A_i| - 2t - 1$ edges. Each T_i is a FST with $|A_i|$ A-points and $|A_i| - 2$ S-points, and therefore has $2 \times |A_i| - 3$ edges, and T must have $2 \times \sum_{i=1}^{t} |A_i| - 3t$ edges. Equating the two expressions for the number of edges in T gives

$$n + \sum_{i=1}^{t} |A_i| - 2t - 1 = 2 \times \sum_{i=1}^{t} |A_i| - 3t,$$

and rearranging yields

$$\sum_{i=1}^{t} |A_i| = n + t - 1.$$

The possible partitions are easily generated by requiring $|A_1| \geq |A_2| \geq \ldots \geq |A_t|$. In this case the maximum size of component A_i is given by $|A_i| = \min(|A_{i-1}|, n + i - 1 - \sum_{j=1}^{i-1} |A_j|)$ where the conditions $\sum_{i=1}^{t} |A_i| = n + t - 1$ and $|A_i| \geq 2 \; \forall \; i$ are always maintained. Table 2.3 shows all the allowable partitions for an eight point set.

For component A_i the following tests are performed before attempting to find its MLFST. The components $A_j \;\; j = 1, \ldots, i-1$ have already been successfully generated and form a partial SMT candidate $T = \bigcup_{j=1}^{i-1} T_j$.

1. The maximum degree of each point in A_i must not be exceeded. If a_j is an A-point in A_i then it can appear in at most $M_j - 1$ of the components $A_1, \ldots, A_{i-1}$ where M_j is the maximum degree of a_j.

[26] See Section 16 Winter [42] or Section 2.1 Hwang *et al.* [19] for references.

t	Allowable Partitions
1	(8)
2	(7,2) (6,3) (5,4)
3	(6,2,2) (5,3,2) (4,4,2) (4,3,3)
4	(5,2,2,2) (4,3,2,2)
5	(4,2,2,2,2) (3,3,2,2,2)
6	(3,2,2,2,2,2)
7	(2,2,2,2,2,2,2)

Table 2.3. All the allowable partitions for an eight point set. The $t = 7$ partition is the partition of the MST.

2. The union $T_i \cup T$ must not contain any cycles. Cycles can be detected immediately if a reachability matrix is used. Although this provides a easy check on cycles the reachability matrix must be updated every time a tree is added or removed from T.[27]

   An alternative test that detects cycles, but not necessarily immediately, is given by the following two conditions:

$$|A_i \cap A_j| \leq 1 \quad \forall\, j = 1, \ldots, i-1$$

$$\sum_{j=1}^{i} |A_j| - \left| \bigcup_{j=1}^{i} A_j \right| < i.$$

   If either condition is not satisfied then a cycle exists.[28]

   The derivation of the second condition is similar to the derivation of the condition on sizes of allowable partitions. Each T_j is a tree with $|A_j|$ A-points and $|A_j| - 2$ S-points. Therefore the union $\bigcup_{j=1}^{i} T_j$ has at most $\sum_{j=1}^{i} (2|A_j| - 3)$ edges. But $\bigcup_{j=1}^{i} T_j$ contains $|\bigcup_{j=1}^{i} A_j|$ A-points and $\sum_{j=1}^{i} (|A_j| - 2)$ S-points and must have $|\bigcup_{j=1}^{i} A_j| + \sum_{j=1}^{i} (|A_j| - 2) - 1$ edges. Therefore

$$\left| \bigcup_{j=1}^{i} A_j \right| + \sum_{j=1}^{i} (|A_j| - 2) - 1 \geq \sum_{j=1}^{i} (2|A_j| - 3),$$

   expanding gives

$$\left| \bigcup_{j=1}^{i} A_j \right| + \sum_{j=1}^{i} |A_j| - 2i - 1 \geq 2 \sum_{j=1}^{i} |A_j| - 3i,$$

   and finally rearranging and changing to a strict inequality by removing the minus 1 gives the condition above.

3. If $|A_i| = 2$ then the two points in A_i must be adjacent in the MST of $\mathcal{A}$.

[27] The reachability matrix of T is a $n \times n$ matrix where $n = |\mathcal{A}|$ with element (i, j) equal to 1 if there is a path from a_i to a_j in T or 0 if there is no path.

[28] It is possible that $T \cup T_i$ is not connected.

4. If $|A_i| \geq 3$ then the Steiner polygon of A_i must not be degenerate.

To find the MLFST of component A_i it is necessary to find all the FSTs and therefore process all the non-equivalent associations (see Section 2.9). All permutations of the $|A_i|$ A-points must be considered, and for all these permutations all possible bracketings of the points must be investigated. The number of permutations that must be considered is reduced by using properties of the Steiner polygon of $\mathcal{A}$. The following important result provides the reduction:

> The association giving the MLFST of A_i is such that the order of A-points on the SP of A_i is identical to the cyclic order of the A-points induced by the SP of A_i. (The A-points not on the SP can be in any position in the association.) Therefore only permutations with the correct cyclic ordering of A-points on the SP of A_i must be generated.[29]

A further reduction can be achieved by having an A-point not on the SP as the last point of the permutation. For component A_i the number of permutations is reduced from $(|A_i| - 1)!$ to $\frac{(|A_i|-1)!}{m!}$ where $m < |A_i|$ is the number of A-points on the SP of A_i (when $m = |A_i|$ the number of permutations is one). For example, Figure 2.32 shows the Steiner polygon for a set of six points. There are four points on the SP. The permutations that must be considered where point 5 is arbitrarily selected from the A-points not on the SP to be the end point of the associations are: $(6, 1, 2, 3, 4, 5)$, $(1, 6, 2, 3, 4, 5)$, $(1, 2, 6, 3, 4, 5)$, $(1, 2, 3, 6, 4, 5)$ and $(1, 2, 3, 4, 6, 5)$. For each of these permutations the non-equivalent associations are generated and an attempt made to find the FST for each association. In this six point example the number of permutations is dramatically reduced from $5! = 120$ to $\frac{5!}{4!} = 5$. The more points on the SP the better!

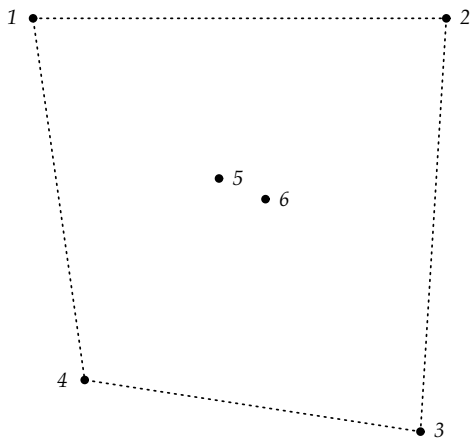


Figure 2.32. An example Steiner polygon for demonstrating the cyclic ordering of A-points induced by the polygon. Only six permutations must be considered for this example.

To generate non-equivalent associations from permutations *pairing vectors* are used. The pairing vector V for a component of size n has $n - 2$ elements V_j

[29] Theorem 16-1 Winter [42].

(1,1,1,1)	(3,3,1,1)
(2,1,1,1)	(3,3,2,1)
(2,2,1,1)	(4,1,1,1)
(2,2,2,1)	(4,2,1,1)
(3,1,1,1)	(4,2,2,1)
(3,2,1,1)	(4,3,1,1)
(3,2,2,1)	(4,3,2,1)

Table 2.4. The pairing vectors for a six point permutation.

j	A^{j-1}	V_j	$A^{j-1}_{V_j}$	$A^{j-1}_{V_j+1}$	A^j
1	$3,6,1,2,5,4$	3	1	2	$3,6,(1,2),5,4$
2	$3,6,(1,2),5,4$	2	6	$(1,2)$	$3,(6,(1,2)),5,4$
3	$3,(6,(1,2)),5,4$	1	3	$(6,(1,2))$	$(3,(6,(1,2))),5,4$
4	$(3,(6,(1,2))),5,4$	1	$(3,(6,(1,2)))$	5	$((3,(6,(1,2))),5),4$

Table 2.5. An example of using a pairing vector to transform a permutation into an association. Pairing vector $V = (3,2,1,1)$ transforms permutation $(3,6,1,2,5,4)$ into association $((3,(6,(1,2))),5),4$.

where $V_j \leq n-j-1 \; \forall \; j$ and $V_j \geq V_{j+1} \; 1 \leq j < n-2$. The pairing vector is used in an iterative fashion to transform a permutation into an association. The initial "association" A^0 is the permutation. The j^{th} association is generated by replacing the V_j^{th} and $(V_j+1)^{th}$ elements of A^{j-1} with the E-point $(A^{j-1}_{V_j}, A^{j-1}_{V_j+1})$, the new association is A^j. The required association is A^{n-2}. The number of non-equivalent associations of a permutation of size n is $\frac{(2n-4)!}{(n-2)!(n-1)!}$.[30] Table 2.4 shows the fourteen pairing vectors for $n = 6$. Table 2.5 shows the transformation of permutation $(3,6,1,2,5,4)$ into association $((3,(6,(1,2))),5),4$ using the pairing vector $V = (3,2,1,1)$.

For each non-equivalent association of each permutation the FST is found, if it exists. The FST with the shortest length, if any, is the MLFST T_i of A_i. The MLFST must satisfy the following conditions if it is to be part of a candidate for the SMT:

1. The length of the tree $\bigcup_{j=1}^{i} T_j$ must be less than the length of the shortest candidate for the SMT found to date.

2. A_i may have a point in common with at least one of $A_1, \ldots, A_{i-1}$. The angle made by edges incident to a common point must be greater than or equal to 120°. More precisely, for some $j < i$ $A_j \cap A_i = \{a_k\}$, in both T_j and T_i there is an edge incident on a_k. The edges must make an angle of at least 120° with each other. (In each tree there is only one edge incident on a_k because the trees are FSTs.)

A tree $T = \bigcup_{j=1}^{i} T_j$ that passes all the tests and is made up of components such that $\bigcup_{j=1}^{i} A_j = \mathcal{A}$ is the new SMT candidate.

[30] Equation 16-4 Winter [42].

For non-degenerate problems up to 12 point problems can be solved in a reasonable time. The time required is strongly dependent on the number of points on the Steiner polygon. The more points on the SP then the fewer permutations to be considered for each component.

2.10.2 Winter's GEOSTEINER algorithm

An alternative to Cockayne's algorithm is GEOSTEINER developed by Winter [42] [43].[31] Some properties of Cockayne's algorithm are undesirable and are improved upon by Winter. These undesirable properties are:

- The computation of the minimum length FST for a component is generally repeated many times;
- S-points must be calculated for every FST;
- Only a small number of geometrical tests are used to reduce the amount of computation. The tests used are the angle test to check the angle between edges of MLFSTs at points in common, and the check on interior angles of the Steiner polygon to determine the maximum degree of A-points in the SMT;
- All associations of the same component are processed at one time. However associations of different components may have common parts and could therefore be processed at the same time. Unfortunately the algorithm does not recognise this and the same E-points are constructed repeatedly.
- The execution time of a problem is strongly dependent on the number of points on the Steiner polygon.

Clearly there is a substantial amount of repeated and unnecessary computation. Winter eliminates as much computation as possible by employing an almost reverse approach compared to Cockayne. Partition, divisions, components are not the language of GEOSTEINER, instead proper associations, dominating edges and points become part of the vocabulary. GEOSTEINER has two distinct parts. The first involves finding all FSTs that can possibly be part of the SMT by using powerful geometrical tests to eliminate E-points and FSTs that can not be part of the SMT. The second part is searching the list of FSTs and forming feasible unions and finding the shortest feasible union, the SMT.

The first part is by far the most complicated. All FSTs are found and tested. This is done once for each FST and therefore each association is processed once. Winter uses *proper associations* to generate non-equivalent associations, and further the order of generation is such that associations with common

[31] Winter [42] is a Master's thesis and Winter [43] is a paper published about four years later. The former is very detailed in description, discussion and justification, while the latter is brief in comparison but describes a more streamlined version of the algorithm.

E-points are processed at the same time. An association is an E-point connected to a terminating A-point. The E-point can be represented as a binary tree of other E-points or A-points. When connected to the terminating A-point an extended binary tree is created. A proper association is an association that when represented as an extended binary tree has a specified well defined structure which allows only a certain labelling of the A-point leaves of the tree. As E-points are formed (this is part of the proper association generation) they and the proper associations are subjected to a number of complex geometrical tests using both properties of the A-points on which they are based and other A-points. The tests are used to determine if the E-point is part of a non-existent FST or the FST can not be part of the SMT. The second part of GEOSTEINER is the processing of the list of successful FSTs. A standard backtracking disjoint set union procedure is used to find the SMT.

GEOSTEINER significantly reduces the computation through the proper association generation mechanism and efficient geometrical tests. The number of E-points and FSTs surviving the tests is typically very small. E-points are only constructed once and S-points are only determined after the SMT has been found by testing feasible unions of FSTs. The execution time of GEOSTEINER is far less dependent than Cockayne's algorithm on the number of points on the Steiner polygon. In fact the efficiency of the tests increases as the number of points on the SP decreases. These features make GEOSTEINER considerably faster than Cockayne's algorithm, and bring the solvable range up to fifteen points.

2.10.3 Cockayne and Hewgill's improvements to GEOSTEINER

Cockayne and Hewgill suggest improvements to the construction of feasible unions part of GEOSTEINER. The first set of improvements nudges the solvable range up to seventeen points (see Cockayne and Hewgill [6]).[32] The changes involve preprocessing of the list of FSTs and the use of decomposition theorems to partition the list into smaller lists that each provide a SMT for a subset of $\mathcal{A}$. The second set of improvements provides more substantial gains (see Cockayne and Hewgill [7]). The gains are based on further preprocessing of the list of FSTs including removal of FSTs, application of the decomposition theorems and a more sophisticated search of the list or lists if decomposition occurred. The solvable range in reasonable time is extended to 30–40 points, although some 100 point problems have been solved.[33]

2.10.4 Proposed algorithms

The proposed *Luminary* algorithm of Hwang and Weng does not find the SMT.[34] Instead it finds the unique Steiner tree in the set of degenerate Steiner topologies

[32] About 80% of their random 30 point problems were solved as long as decompositions of no more than 17 points were obtained.

[33] Appendix B contains thirty 100 point test problems. These problems are used in later chapters to test the success or otherwise of heuristics for the Euclidean Steiner tree problem.

[34] Section 2.8 Hwang *et al.* [19].

of full Steiner topology T. This set is denoted by $D_S(T)$. A degenerate topology of T is obtained from T by deleting edges and collapsing the end points of the edges. To find the SMT all full Steiner topologies must be investigated. The time to find the unique Steiner tree of $D_S(T)$ is $O(n^2)$. Unfortunately the number of full Steiner topologies for n points is a super-polynomial function of n (see Table 2.1).

The proposed *Negative Edge* algorithm of Trietsch and Hwang is similar to the Luminary algorithm in that it seeks to find the unique Steiner tree for every $D_S(T)$.[35] It uses lower bounds on the lengths of FSTs and the branch and bound technique to construct a SMT.

[35]Section 2.7 Hwang *et al.* [19] and Trietsch and Hwang [39].

Chapter 3

Selected Applications of the Euclidean Steiner Tree Problem

Three applications of the Euclidean Steiner tree problem are described in this chapter. The necessary generalisations of the problem are discussed and either optimal or approximate methods are presented. The Euclidean Steiner tree problem is an appropriate starting point from which to consider the problems. The generalisations are understandably more difficult to solve than the Euclidean Steiner tree problem. Hence optimal solutions to all but the smallest problems are impossible to find. Regardless of the success or otherwise of the methods, the applications demonstrate that the Euclidean Steiner tree problem is a fundamental problem of network design from which many real problems are derived.

3.1 Augmenting an Existing Network

Trietsch [40][41] investigates generalisations of the Euclidean Steiner tree problem. The two problems are:

1. Given an existing network and a collection of points, connect the points to the network at either vertices or arcs of the network by links of minimal length.

2. Given many existing networks connect the networks using minimal length links.

Both problems are of considerable practical importance. The first is applicable to the problem of connecting new customers to an electrical network. The second is a generalisation of the first problem and is applicable to the following problems:

- The connection of new computers and communication systems to an existing network.

- The interconnection of several sewerage systems to a common treatment plant.

- The interconnection of electricity supply networks to better cope with peak time demands and provide backup in times of failure.

A generalisation of the Euclidean Steiner tree problem is an option for such problems if it is assumed the cost of making a connection to an existing network and the cost of junctions is negligible, and the Euclidean distance is a good approximation to the cost of a link. This ignores that cost may be dependent upon flow along a link and the world is not flat. The standard Euclidean Steiner tree problem related decision problem is NP-hard, therefore the generalisations are also NP-hard.

In the remainder of this section extensions of some fundamental properties of Steiner minimal trees and Steiner polygons are described, and these are used to aid the solution of an example of the problem of connecting a set of points to an existing network.[1]

3.1.1 The problem

The existing network is $G(V, A)$ where V is the set of vertices and A is the set of arcs spanning V, N is the set of points to be connected to G using minimal length links. The optimal connection of G and N gives a *generalised Steiner minimal tree*, or *GSMT*. G can be any network, therefore the GSMT isn't necessarily a tree. If G contains cycles then so too will the GSMT. The GSMT for G and N will not always be a Steiner tree for the set of points $N \cup V$. Figure 3.1 shows a network with vertices $A = \{A, B, C, D\}$ and ten points that must be connected.

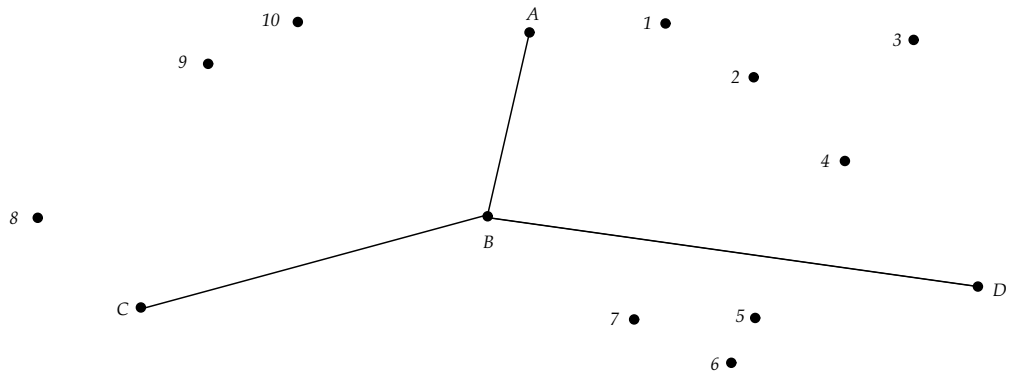


Figure 3.1. The example problem: a network and the ten points which must be connected to it using minimal length links.

Consider a network consisting of two vertices and one arc, a segment. The connection of another point to the network will be either by a line to one of the endpoints, or by a line perpendicular to the segment. This is called the *basic case* (see Figure 3.2).

If $|N| > 1$ then the GSMT will be made up of one or more SMTs of subsets of N connected to the segment or its endpoints. This extends to a network with

[1] This is the subject of Trietsch [40]. The example problem used by Trietsch is used here to demonstrate important properties and the method of solution.

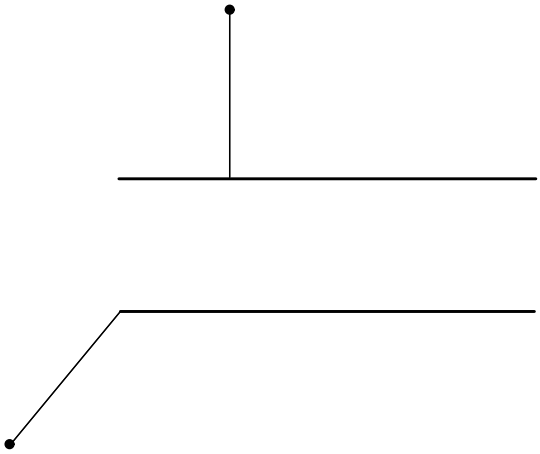


Figure 3.2. Connecting a point to a segment using the smallest length connection.

more than two vertices and many arcs. A GSMT where N is connected to G by one link is called a *simple* GSMT. Any other GSMT is a *compound* GSMT, and is a combination of simple GSMTs for subsets of N and A (see Figure 3.3).

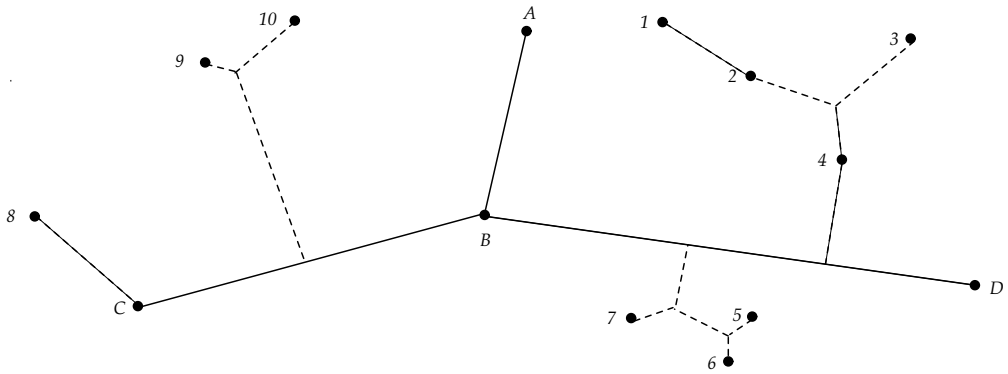


Figure 3.3. The GSMT for the example problem is the connection of four simple GSMTs. Three by perpendicular links and one by connection to an endpoint.

A possible approach to finding the GSMT of G and N is to consider all subsets M of N in combination with each arc of G and find the simple GSMT of each combination. Each subproblem involves finding the SMT for $|M| + 1$ points where the arc is considered a "super-point". The tree spanning M is joined to the arc either by a line to an endpoint or by a perpendicular to the segment. The connecting link may originate from a Steiner point or a given point of the GSMT. For example, in Figure 3.3 the GSMT of $\{1, 2, 3, 4\}$ and arc BD is connected by a link to 4, while the GSMT of $\{9, 10\}$ and arc BC is connected by a link to the Steiner point. This can be incorporated into a modified method of finding a full Steiner tree, the cornerstone of the SMT algorithm, to give a *generalised full Steiner tree*, or *GFST*. The shortest collection of simple GSMTs spanning $N \cup V$ is the GSMT. However, knowledge of the

ordinary Euclidean Steiner tree problem indicates this approach is intractable. Trietsch offers hope by introducing the *generalised Steiner polygon* or *GSP*, and with ordinary Steiner polygons and convex hulls shows how the search for the GSMT can be improved.

3.1.2 Two decomposition methods

The Steiner polygon of N and existing network G provide a valuable decomposition if the two intersect. The intersections create disjoint sets of N that can be solved separately. This is demonstrated in Figure 3.4. The Steiner polygon intersects G at arcs BC and BD, splitting the problem into two smaller problems for sets of points $\{1, 2, 3, 4, 8, 9, 10\}$ and $\{5, 6, 7\}$.

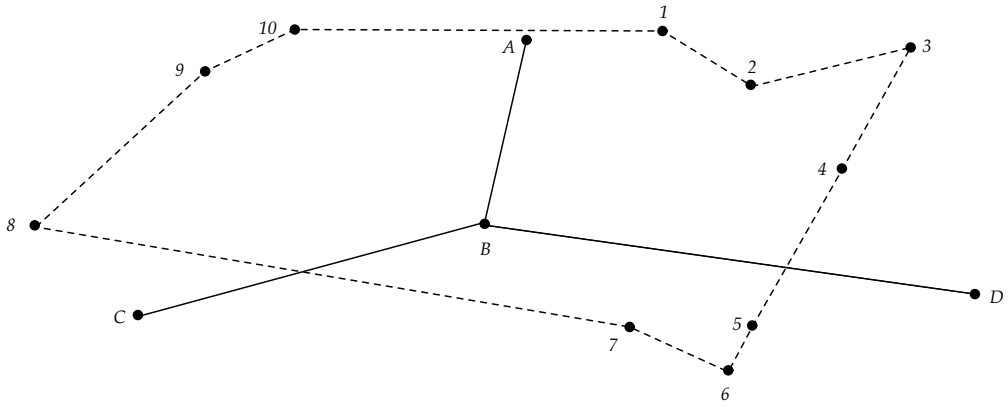


Figure 3.4. The Steiner polygon of $\{1, \ldots, 10\}$ intersects the existing network and decomposes the problem into two smaller problems: $\{1, 2, 3, 4, 8, 9, 10\}$ and $\{5, 6, 7\}$.

The second decomposition method uses the Steiner polygon of $N \cup V$, called Q. If Q can be partitioned by sequences of edges of G then the problem is decomposed into smaller problems with sets of points given by the points in the disjoint sets of the partition of Q and in N. For example, in Figure 3.5 the Steiner polygon of $\{1, \ldots, 10\} \cup \{A, B, C, D\}$ is partitioned by the edges AB, BC and BD into three smaller problems for the sets $\{1, 2, 3, 4\}$, $\{5, 6, 7\}$ and $\{8, 9, 10\}$. This is one better than the first decomposition method.

The second method is more powerful than the first, but the first is easier to grasp and use. Repeated application of the first method does not necessarily give as good a decomposition as the second method. For instance applying the first to $\{1, 2, 3, 4, 8, 9, 10\}$ in the example problem does not give a decomposition.

Once a decomposition, if any, has been found the arcs of G that must be considered with subsets M of N can be determined. The rule for deciding which arcs of G need be considered for connection with M is: only arcs which can be connected to the convex hull polygon of M by a straight line which does not cross any other arc of G need be considered for connection to M. Not all arcs need be considered. For example, in Figure 3.5 arc AB shouldn't be combined with any subset of $\{5, 6, 7\}$, and subsets of $\{1, 2, 3, 4\}$ can be combined with arcs AB and BD.

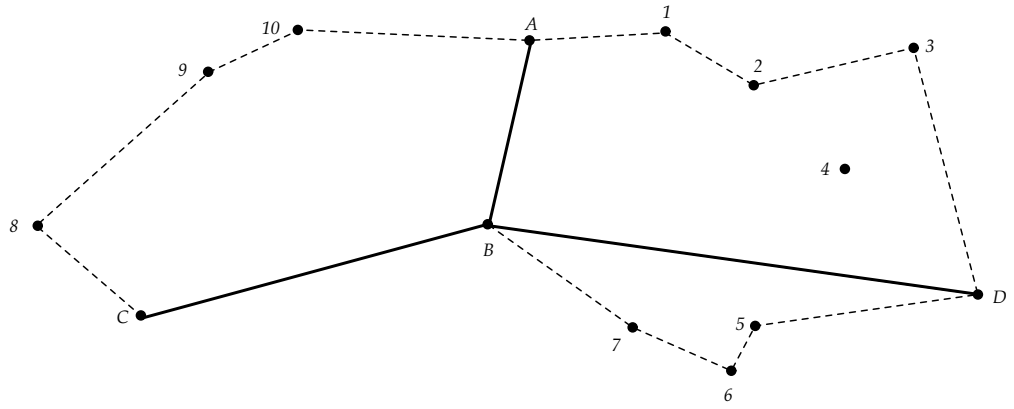


Figure 3.5. The Steiner polygon of $\{1, \ldots, 10\} \cup \{A, B, C, D\}$ is partitioned by edges in G and decomposes the problem into three smaller problems: $\{1, 2, 3, 4\}$, $\{5, 6, 7\}$ and $\{8, 9, 10\}$.

3.1.3 The generalised Steiner polygon

In the standard Euclidean Steiner tree problem the Steiner polygon provides a method of decomposition and gives a cyclic ordering to points which is beneficial in reducing the effort in constructing full Steiner trees and Steiner minimal trees. Two further types of Steiner polygon are defined for the generalised Steiner tree problem: the *semi-generalised Steiner polygon*, or *SGSP*, and the *generalised Steiner polygon*, or *GSP*. Both are useful when constructing GSMTs and have the following properties:

- the GSMT is within the SGSP;
- a *simple* GSMT is within the GSP;
- the cyclic order of points on the GSP can be used in the construction of GFSTs.

The method of finding a SGSP and GSP is discussed for the case of a network $G(V, A)$ having only two vertices and one arc connecting the two vertices by a segment. The vertices are called a and b. The segment is considered a super-point for the purposes of the SGSP and GSP construction. This case is of prime interest because the GSMTs of subsets of points with one arc are the fundamental building blocks of larger GSMT problems.

The first step in finding the SGSP is to join all points to all other points using $\frac{1}{2}(n+1)n$ segments, where $n = |N|$. The segments connecting a point in N to the super-point are done so as in the *basic case*, that is by a perpendicular line to the segment ab or by a line to either a or b. The convex hull polygon of these segments is the starting point for finding the SGSP. Figures 3.6 and 3.7 show the initial step of SGSP construction. In the first figure all of the segment ab is part of the convex hull polygon, in the second the segment cd, a subset of ab, is part of the convex hull polygon.

The next phase of finding the SGSP is identical to the iterative method of the ordinary Steiner polygon construction. For each segment kl of the convex

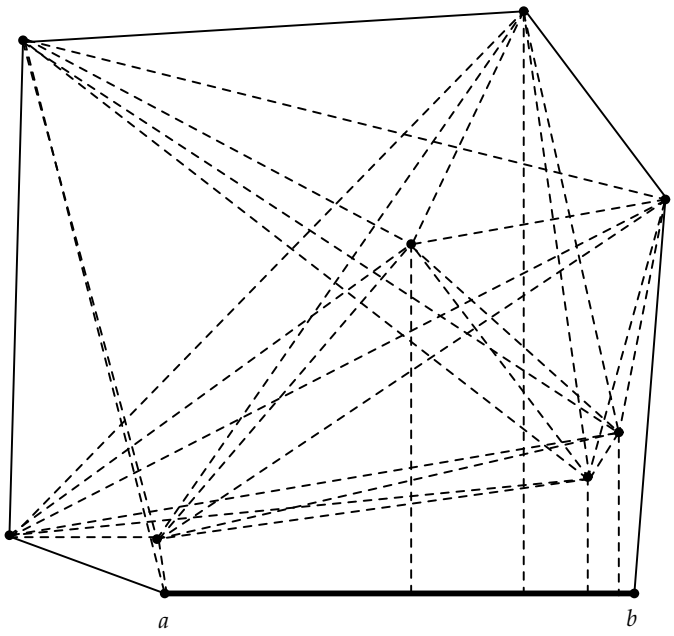


Figure 3.6. The first step in constructing the semi-generalised Steiner polygon: forming the convex hull polygon.

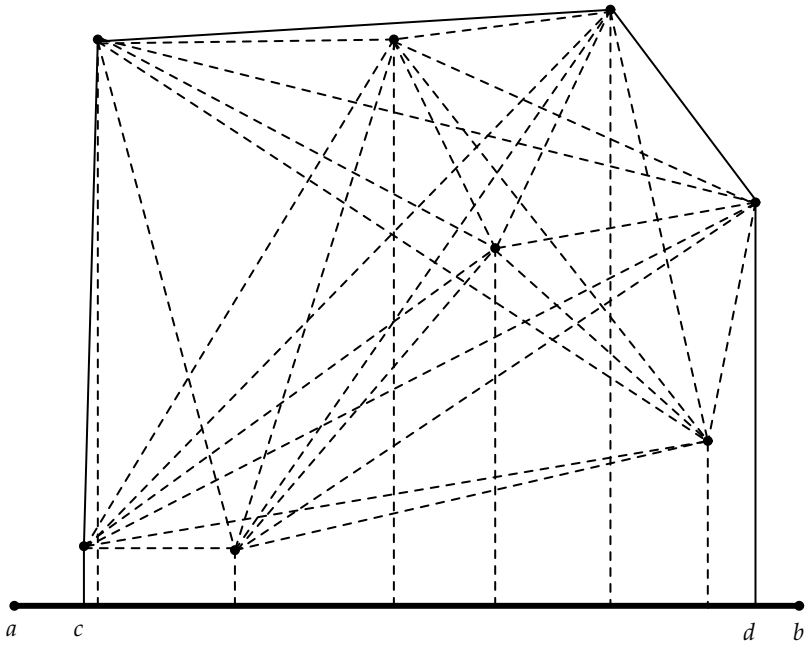


Figure 3.7. The first step in constructing the semi-generalised Steiner polygon: forming the convex hull polygon. Only segment cd of the segment ab is part of the polygon.

hull polygon, where $k, l \in N$, a third point $m \in N \cup \{a, b\}$ is found such that the triangle formed by the three points contains no other points and the angle made at m by the segments km and ml is greater than or equal to 120°. If such an m is found then the edge kl is replaced by km and ml. This gives a new polygon and the step is repeated until no more changes can be made. The final polygon is the SGSP.

Applying this to the convex hull polygon of Figure 3.6 gives no changes. Therefore the convex hull polygon is the SGSP. The SGSP for the points and network shown in Figure 3.7 is shown in Figure 3.8. Only one small change is possible: a point near the polygon at the top of the figure is incorporated into the SGSP. The rightmost interior point can not be added because both k and l must be in N, the point d is not in N.

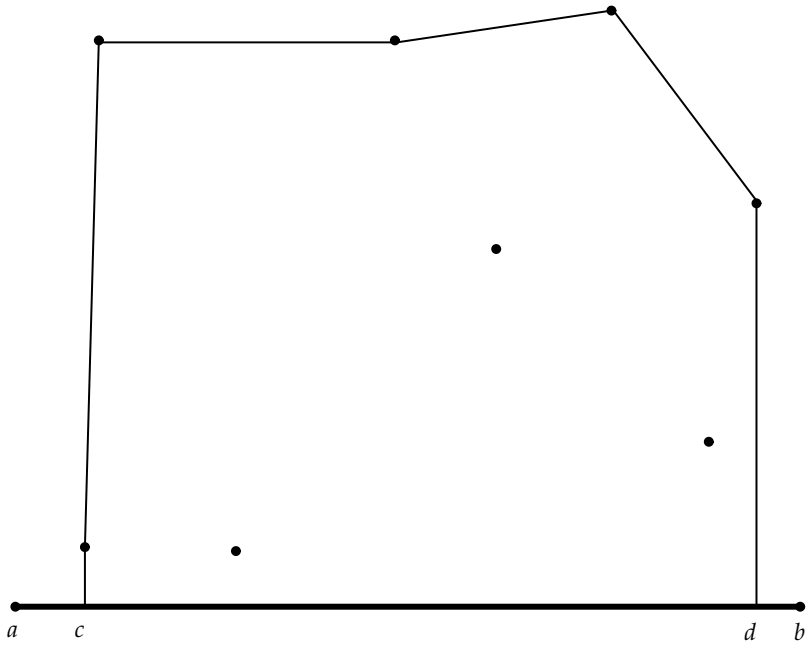


Figure 3.8. The application of the iterative method to the convex hull polygon of Figure 3.7 gives only one change.

The generalised Steiner polygon is found by a similar iterative process starting from the semi-generalised Steiner polygon. For each edge kl, where $k \in N$ and l can be a member of N or any point on the segment ab (or cd), a third point $m \in N$ is found such that the region enclosed by segments kl, km and ml contains no other points and the angle made at m by the segments km and ml is greater than or equal to 120°. If such an m is found then the edge kl is replaced by km and ml. The above is repeated using the new polygon until no further changes are possible. The final polygon is the GSP.

The definition of l allows it to be the super-point, the segment ab (or possibly cd depending on the problem). The segments kl, km and ml do not necessarily form a triangle, the point at which kl meets ab may not be identical to where ml meets ab. The actual point on the segment ab is defined by the *basic case*. This is demonstrated in Figure 3.9 using the SGSP of Figure 3.8 as a starting

point. The new polygon in the figure is the GSP for the problem. The edges kd (equivalent to kl), km and me (ml) define a region containing no other points and the angle at m is greater than 120°.[2] Hence kd is replaced by km and me.

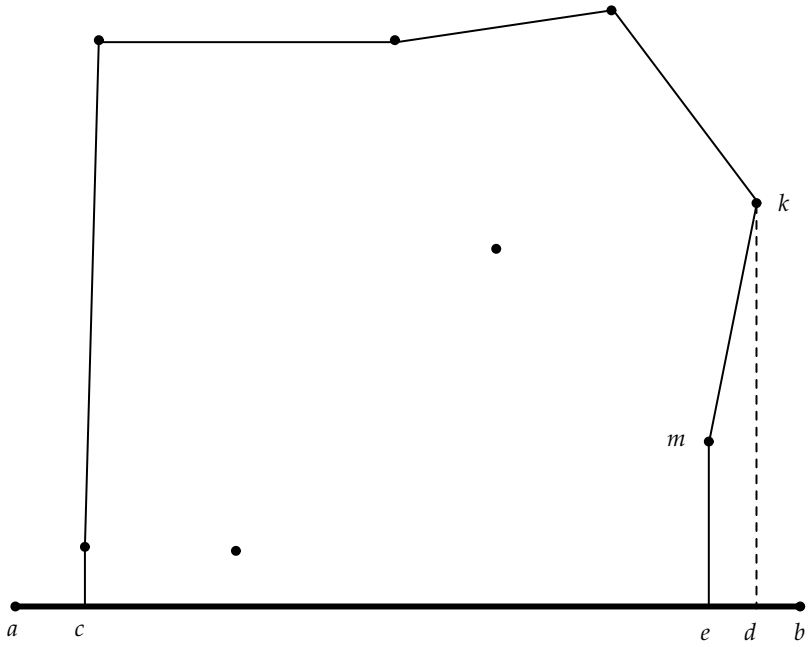


Figure 3.9. The application of the iterative method to the semi-generalised Steiner polygon of Figure 3.8 gives only one change. The dashed edge is removed and replaced by segments km and me.

The GSP of the problem in Figure 3.6 is shown in Figure 3.10. The move from the SGSP to the GSP involves two changes (both on the right hand side of the figure near b).

The decomposition methods, the SGSP and the GSP provide valuable information for the search for the minimal length connection of a set of points to an existing network. Unfortunately the problem still remains very difficult as it involves the solution of many Euclidean Steiner tree problems and is tractable for only small problems and decomposed sub-problems.

3.2 Finding a Minimum Cost Communication Network

The problem of designing a communication network that connects a number of sites where there is a cost per unit distance of transmission medium can be considered a generalisation of the Euclidean Steiner tree problem. Gilbert [15] considers this generalisation. He investigates networks for different forms of

[2]Trietsch is not clear about whether the angle kme or kmd must be greater than or equal to 120°. He states angle kml must be greater than or equal to 120°. A safe approach is to assume kmd is the required angle. Angle kme will always be greater.

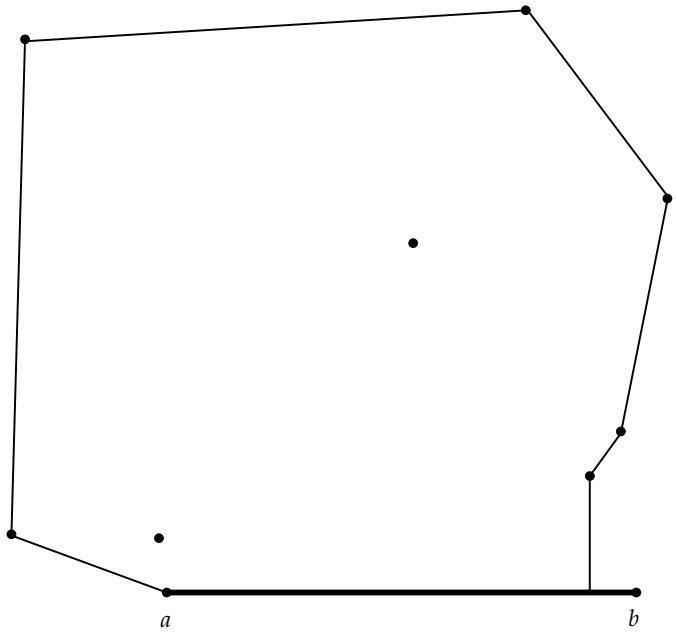


Figure 3.10. The generalised Steiner polygon of the problem in Figure 3.6.

cost function and provides a geometrical technique for finding minimal cost connections.

Gilbert approaches the problem from a Steiner tree perspective by generalising properties of Euclidean Steiner trees. Although for a particular problem the minimal generalised Steiner tree may not be the lowest cost solution, but it will be a very good solution. The minimum cost network may not necessarily be a tree, or if a tree it may contain points that have four or more lines incident on the points.[3] The assumptions for using the Euclidean Steiner tree problem as a starting point are that the world is flat and the cost of a junction is negligible.

3.2.1 The problem

Given a set of sites or cities, $\mathcal{A}$, to be connected by a communication network where $N(i,j)$ communication channels are required between each pair of cities A_i and A_j and the cost per unit distance of providing N channels is $f(N)$. The objective is to find a network with minimal cost, that is minimise $\sum_{l \in G} d_l f(N_l)$ where G is the network, l is a link in G, d_l is the length of link l and N_l is the number of communication channels required on link l.

Special cases of the function $f(N)$ lead to particular forms of minimal connection networks:

- $f(0)$ represents the fixed or preliminary cost of a link and is independent of the number of channels. Examples of such costs are legal fees, surveying costs, the cost of digging a trench or installing microwave transmitters (as costs per unit distance).

[3]The generalised Steiner trees of this section are different to the GSMTs of Trietsch.

- If $f(N)$ is independent of N then the minimum cost network is the Steiner minimal tree of $\mathcal{A}$. In this case the only costs per unit distance are preliminary costs. Using the SMT may be satisfactory approach if preliminary costs far outweigh the costs of the actual transmission medium (as costs per unit distance).
- If $f(N) = N$ then the the minimal network is a complete network. This is a network where every city is connected to every other city by a direct link.
- If $f(N) = a + bN$ then simple bounds can be found for the minimum cost. The additional cost of adding one more channel is b and is independent of N.

3.2.2 Bounds for a linear cost problem

For minimal network G with cost function $f(N) = a + bN$ the total network cost C_G is

$$C_G = a\sum_{l\in G} d_l + b\sum_{l\in G} d_l(N_l).$$

Bounds for the total cost can be derived by considering each component of the expression separately. The total preliminary costs must be at least those of the SMT of $\mathcal{A}$, that is aD_S where D_S is the length of the SMT. The total transmission medium related costs are minimised by using a complete network, this has cost $b\sum_{i<j} d(i,j)N(i,j)$ where $d(i,j)$ is the distance between cities A_i and A_j. A lower bound for C_G is

$$C_l = aD_S + b\sum_{i<j} d(i,j)N(i,j).$$

An upper bound for C_G is the cost of the complete graph

$$C_u = aD_C + b\sum_{i<j} d(i,j)N(i,j),$$

where $D_C = \sum_{i<j} d(i,j)$, the length of the complete network. The bounds can be simplified by defining ν the average number of channels between cities weighted by the distance

$$\nu = \frac{1}{D_C}\sum_{i<j} d(i,j)N(i,j).$$

The bounds for the minimal cost are then

$$aD_S + b\nu D_C \leq C_G \leq aD_C + b\nu D_C\,.$$

The quantity ν can be thought of as a measure of traffic level, and is convenient when the number of channels is only known in a relative sense, for example the number of channels required from A to B is five times the number from B to C. Gilbert states that the lower bound is accurate at low and high values of ν and at worst is about 11% below the true minimal cost for any ν.

3.2.3 A three city minimum network

Understanding the construction method of a simple three city network is fundamental to the construction of larger networks. The construction is shown by example and only Steiner topologies are investigated. The method is not applicable where more than three links can be incident on a point. The main difference between this method and the ordinary Steiner tree method is that equilateral triangles are no longer used. Instead triangles with edge lengths dependent on costs per unit distance are used to find "equilateral points" and arcs on which "Steiner points" lie.

The three point generalised Steiner tree with one Steiner point can be found using a ruler and compass. The general case for three points A_1, A_2 and A_3 with respective costs of c_1, c_2 and c_3 to the S-point is demonstrated. Using any two points, A_1 and A_2 in this case, their E-point $A_{1,2}$ is the point which is distance $d_1 = \frac{dc_2}{c_3}$ from A_1 and distance $d_2 = \frac{dc_1}{c_3}$ from A_2, where d is the distance from A_1 to A_2. $A_{1,2}$ is on the opposite side of the line passing through A_1 and A_2 from A_3. The S-point is the intersection of the segment from $A_{1,2}$ to A_3 and the arc A_1A_2 of the "E-circle" that circumscribes the three points A_1, A_2 and $A_{1,2}$. If the S-point does not exist then the generalised Steiner tree with the given topology does not exist. Figure 3.11 is an example of the construction.

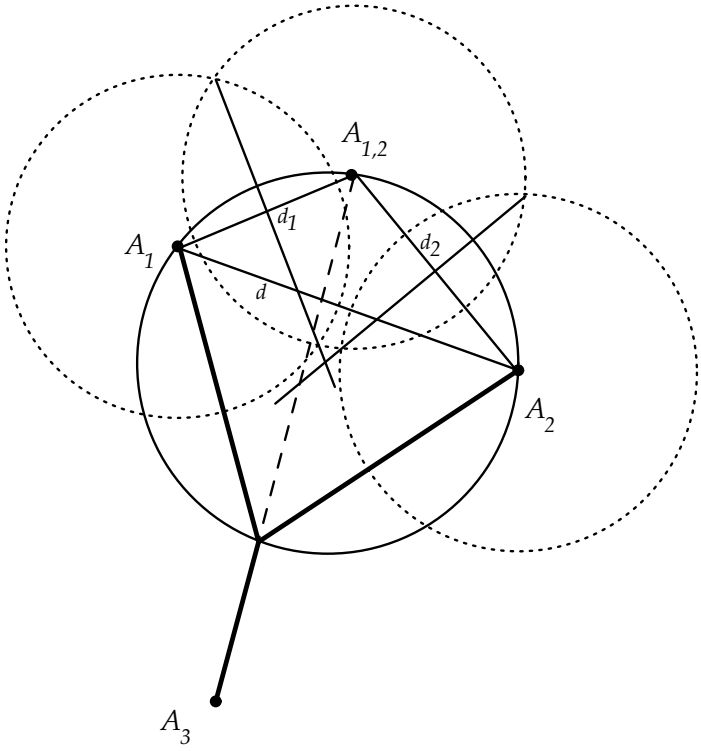


Figure 3.11. The construction of a three point generalised Steiner tree with one Steiner point. The dotted circles are used to find the center of the circumscribing circle. The generalised Steiner tree is shown with bold lines. In this construction the cost per unit distance of the link to A_3 is higher than either of the other two link costs and has pulled the Steiner point closer to A_3.

The example problem is to connect three cities A, B and C at coordinates (2,3), (4,8) and (9,6) respectively. The communication channel requirements are 10 between A and B, 3 between A and C and 2 between B and C. The cost per unit distance function is $f(N) = 100 + 6N$. The SMT for the cities is shown in Figure 3.12 together with the channel requirements for a network with the same topology. The link costs per unit distance are 172, 178 and 130 for the links from the S-point to A, B and C respectively. The S-point in the SMT is at (4.6340,6.5207). The length of the SMT is 10.4033 and has cost 1631. The length of the complete network is 18.3861 and has cost 2040 (which is determined below as the upper bound of the minimum cost network).

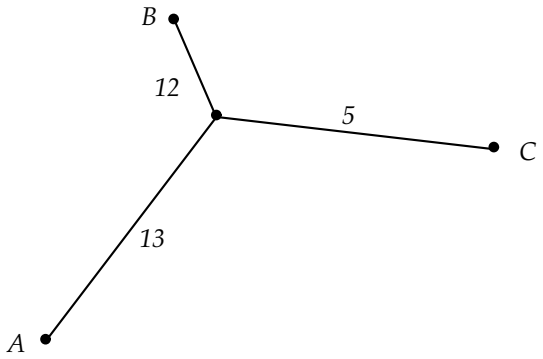


Figure 3.12. The SMT for the three city problem and the channel requirements for the topology.

Table 3.1 shows the complete network distances and the weighted channel requirements. The value of ν is $\frac{33.6170}{18.3862} = 1.83$. Using this and other quantities bounds for the cost of the minimum cost network are determined. The lower bound is $100 \times 10.4033 + 6 \times 1.83 \times 18.3862 = 1242$ and upper bound $100 \times 18.3862 + 6 \times 1.83 \times 18.3862 = 2040$.

Link	Distance	Channels	Weighted Channels
(A, B)	5.3852	10	53.852
(A, C)	7.6158	3	22.847
(B, C)	5.3852	2	10.770
	18.3862		33.6170

Table 3.1. Distance and channel requirements for the three city problem.

The generalised Steiner tree with the same topology as the SMT above can easily be found, if it exists, using the method described above. Using B and C, their E-point is $\frac{5.3852 \times 130}{178} = 3.9330$ from B and $\frac{5.3852 \times 172}{178} = 5.2037$ from C. The generalised Steiner tree construction is shown in Figure 3.13. This network is slightly different to the SMT. The S-point is pulled towards A and B because of the relatively higher costs per unit distance of the links connecting the points.

The S-point is at (4.2912,6.6349). The cost of this generalised Steiner tree is 1623, about 0.5% cheaper than the SMT.

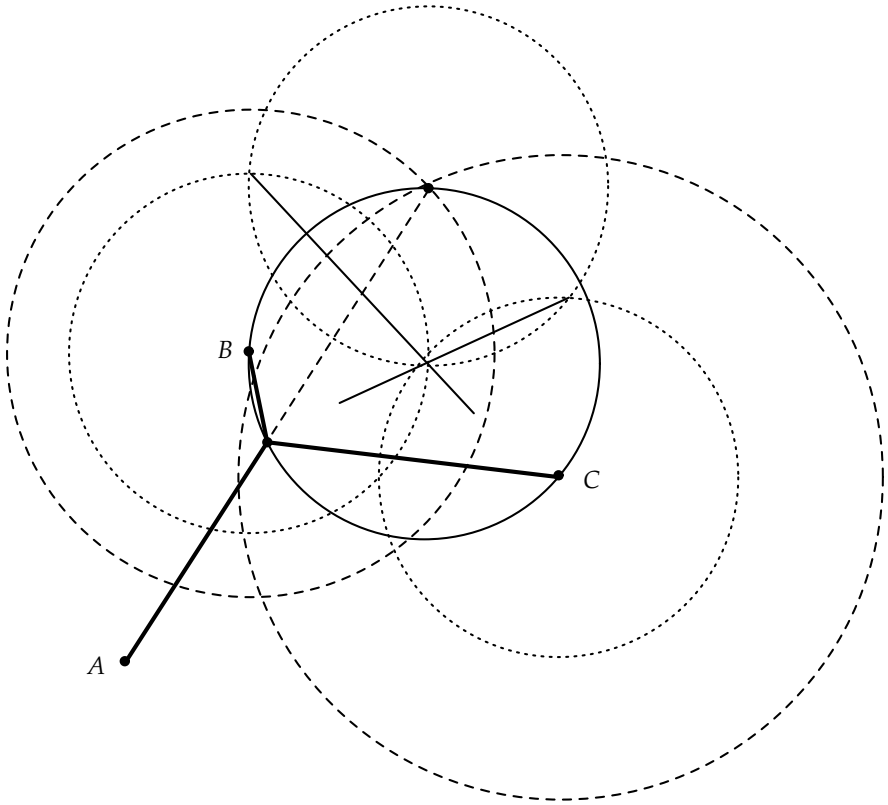


Figure 3.13. The example three city problem generalised Steiner tree construction. The dashed circles are used to find the E-point of B and C. The dotted circles are used to find the center of the circumscribing circle.

There are three other possibilities to consider. Each has no S-point and has two links. These are shown in Figure 3.14. The total cost of the first network is $(100 + 6 \times 13) \times 5.3852 + (100 + 6 \times 5) \times 5.3852 = 1659$. Similar calculations give costs of 2282 and 1961 for the second and third networks respectively. The complete network is cheaper than the second! The ten channel requirement for A to B is needlessly routed through distant C.

3.2.4 Minimum cost networks for four or more cities

For four or more cities the generalised Steiner tree with a given topology is found using the same reduction and expansion techniques presented in Chapter 2. Instead of creating E-points and E-arcs using equilateral triangles and circles circumscribing the equilateral triangles, the triangles with edge lengths based on the relative costs of links are used (as demonstrated in the three city problem above).

A five city problem is solved using the ruler and compass method. The five cities have channel requirements given in Table 3.2, and Figure 3.15 shows the

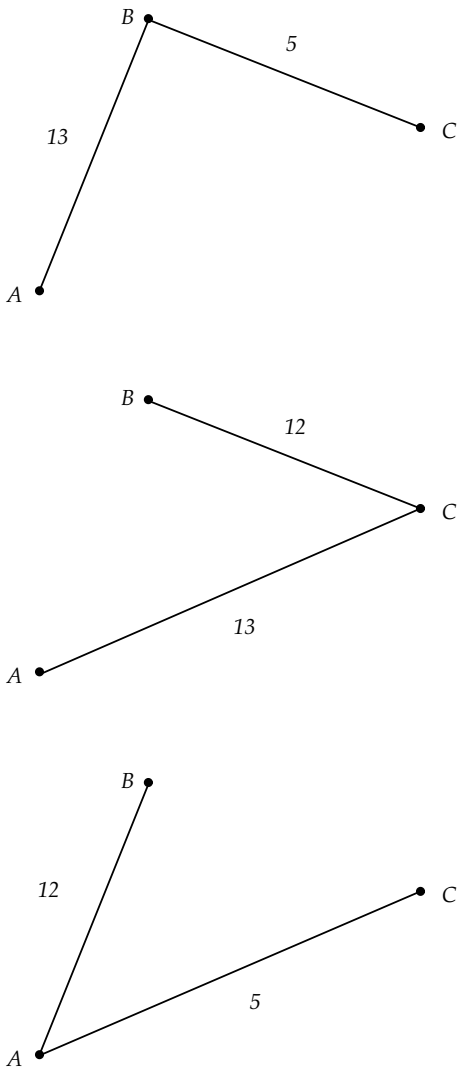


Figure 3.14. The three possible two link networks and the requirements on each link.

location of the cities, the SMT and the link requirements for the SMT topology.

City	B	C	D	E
A	10	3	1	5
B		2	12	2
C			6	4
D				10

Table 3.2. Channel requirements for the five city problem.

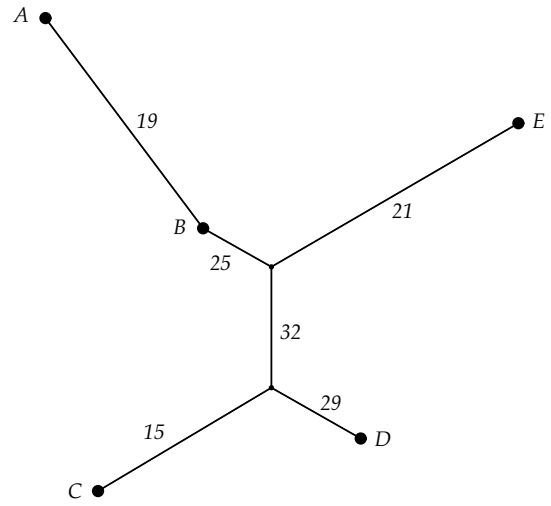


Figure 3.15. The example five city problem SMT and channel requirements for the topology.

To find the generalised Steiner tree with the same topology as the SMT it is necessary to find the E-points corresponding to B and E, and C and D using the relevant costs per unit distance. The two S-points are the intersections of the segment joining the E-points and the E-circles. The ruler and compass construction is shown in Figure 3.16 where a $50 + 5 \times N$ cost per unit distance function is used. The two S-points have been pulled towards D and the link joining the two is shorter than in the SMT. The D link and S-points' links have the highest channel requirements. The reductions in length are at the expense of the B, C and E links, which have lower channel requirements. It is important to note that this network is not necessarily the minimal cost network. There are many more Steiner topologies to be investigated.

The minimal cost network is not necessarily a generalised Steiner tree and may in fact have cycles or have four links incident on a point. The Steiner methods will only find Steiner trees if they exist. And to find the minimal generalised Steiner tree for a problem it is necessary to investigate all feasible Steiner topologies. This has been shown in the previous chapter to be an almost impossible task for all but the smallest problems.

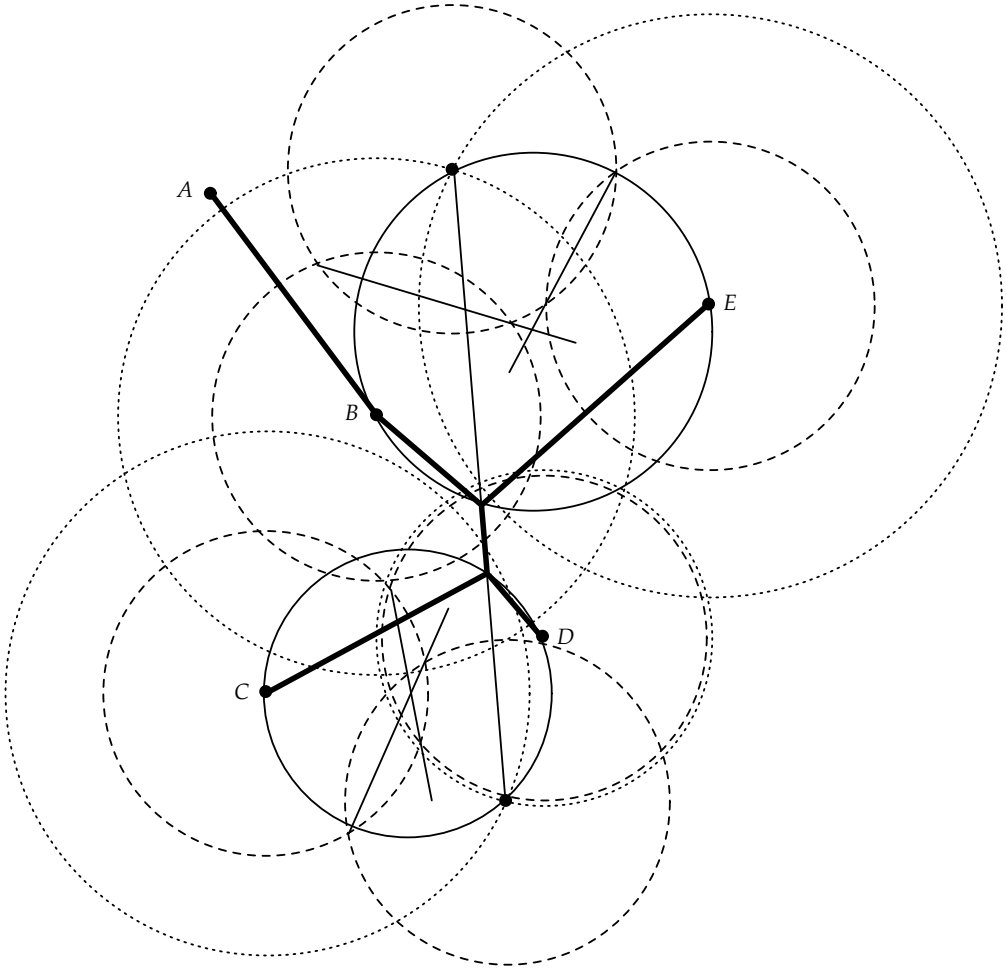


Figure 3.16. The example five city problem generalised Steiner tree with the same topology as the SMT. The bold lines show the generalised Steiner tree, the dotted circles are used to find the E-points, the dashed circles are used to find the centers of the circumscribing circles, and the solid circles are the E-circles.

3.3 Building Design using Steiner Trees and Steiner Circuits

Smith and Liebman [35] investigate the layout of services within buildings and the formulation of the layout problems as either Euclidean Steiner tree problems or rectilinear Steiner tree problems. *Steiner circuits* are introduced as possible solutions to certain classes of layout problems. The services of a building include corridors, stairways, elevators, heating, ventilation, air conditioning, plumbing, lighting and other electrical systems. If the lengths of the service networks are of prime concern and the services represent a significant portion of the cost of a building then finding optimal length configurations is of importance.[4]

Both the Euclidean and rectilinear Steiner problems are undirected and without costs. For services such as plumbing and drainage flow is only possible in one direction. These services and heating, ventilation and air conditioning have links in the network with different costs depending on the points at which the links begin and finish: different points, or rooms, will have different requirements. Smith and Liebman describe approximate methods using directed Steiner trees and weighted directed Steiner trees.[5]

Steiner circuits are networks that visit each given point once and only once, there are no direct connections between given points and each given point is connected to a point somewhere along the segment joining the given point to its adjacent Steiner point in a Steiner tree.[6] Circuits rather than trees can be used for services in a building. Circuits tend to minimise the sum of the shortest paths between all pairs of points, while trees simply minimise the total service network length.

Figure 3.17 shows a Steiner circuit for six points. The position of the points along the segments joining the given points to their adjacent Steiner points are determined by the flow costs for different links. The underlying Steiner tree for the Steiner circuit is the limiting case of a circuit as the flow costs decrease to zero. The Steiner tree provides a lower bound for the Steiner circuit in a similar manner that the minimum spanning tree provides a lower bound for the optimal travelling salesman tour. Smith and Liebman give an iterative algorithm for finding a Steiner circuit for general flow and construction costs. The algorithm solves a travelling salesman problem at every iteration with added constraints to disallow direct connections between given points.

[4]Other possible objectives are the minimisation of the number of intersections or bends in the service networks. Flow characteristics should also be considered for some services. Smith and Liebman suggest finding an optimal topology first, then considering the size and scale of service second. To solve both problems simultaneously is too large a task.

[5]Directed trees with a node to which no edge is directed are called *arborescences*. The node to which no edge is directed is called the *root node*.

[6]This definition implies the Steiner tree is a full Steiner tree for the points and the tree exists. Smith and Liebman's algorithm for a Steiner circuit begins with the SMT, this is not necessarily a full Steiner tree.

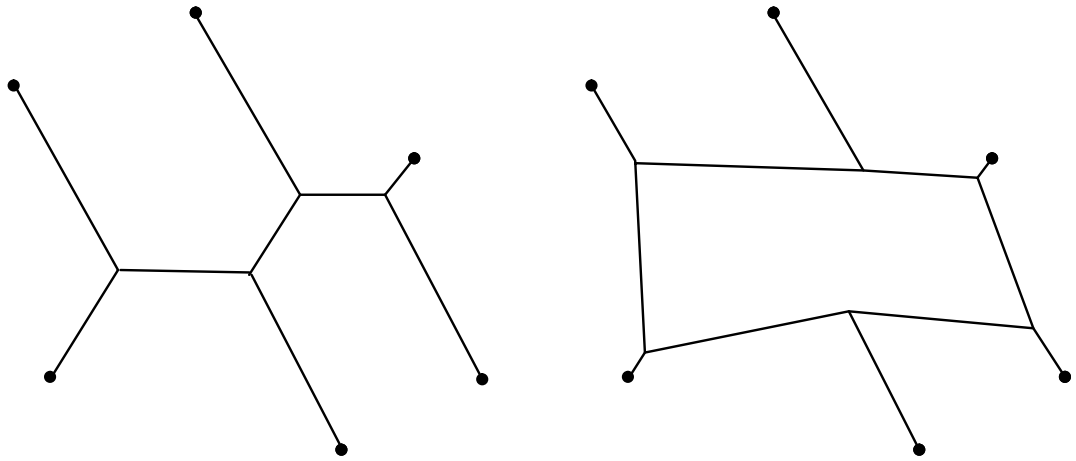


Figure 3.17. On the left is a six point FST, on the right is a Steiner circuit based on the FST. The relative flow costs determine the location of junctions on the segments joining the given points to the Steiner points of the FST.

Smith and Liebman also describe a heuristic for the Euclidean Steiner tree problem which is further improved by Smith *et al.* [37]. The improved heuristic is described and demonstrated by example in Chapter 4.

Chapter 4

Heuristics for the Euclidean Steiner Tree Problem

The inherent intractability of the Euclidean Steiner tree problem has encouraged the development of methods to find approximate Steiner minimal trees. These methods, or *heuristics*, are designed to find good connections of points in times that are generally low order polynomials of problem size. There is a trade-off between the quality of a solution and time spent finding the approximate solution. In this chapter two heuristics for the Euclidean Steiner tree problem are described in detail. Other heuristics are briefly described in the following section.[1]

4.1 An Introduction to Heuristics for the Euclidean Steiner Tree Problem

The simplest heuristic solution to a Euclidean Steiner tree problem is the minimum spanning tree. The MST is relatively easy to find in $O(n \log n)$ time and its length is guaranteed to never exceed the length of the SMT by about 15.5%. Improving upon the MST is at the heart of many methods of approximate solution.

A naïve heuristic based on improving the MST is to add a Steiner point, or S-point, to sets of three points connected by two edges in the MST, and remove the MST edges and replace them with edges to the S-point. The pairs of edges in the MST must share a common end point and a reduction in length must be achieved by introducing the S-point. This requires that the largest angle in the triangle formed by the three points is less than 120°. Further, cycles must be avoided when replacing edges. A cycle results if two sets of three points have two points in common. To overcome this difficulty and attempt to get as large a reduction as possible in total length the sets of three points are considered in decreasing order of reduction offered by introducing a S-point, and sets that do not give rise to a reduction are not considered. An example of this heuristic

[1] Chapter 4 Hwang *et al.* [19], Hwang and Richards [17] and Beasley and Goffinet [2] give introductions to the heuristics for the problem. The first reference is the most extensive.

is shown in Figure 4.1. The order of the sets of points is A, B, E, D and C. The two pairs of sets of points B, C and C, D both share two points. To process set C, after B and D, creates a cycle and is therefore not done. The tree obtained after adding the S-points for the first four sets in the ordered list is the heuristic solution, and is shown in Figure 4.2.

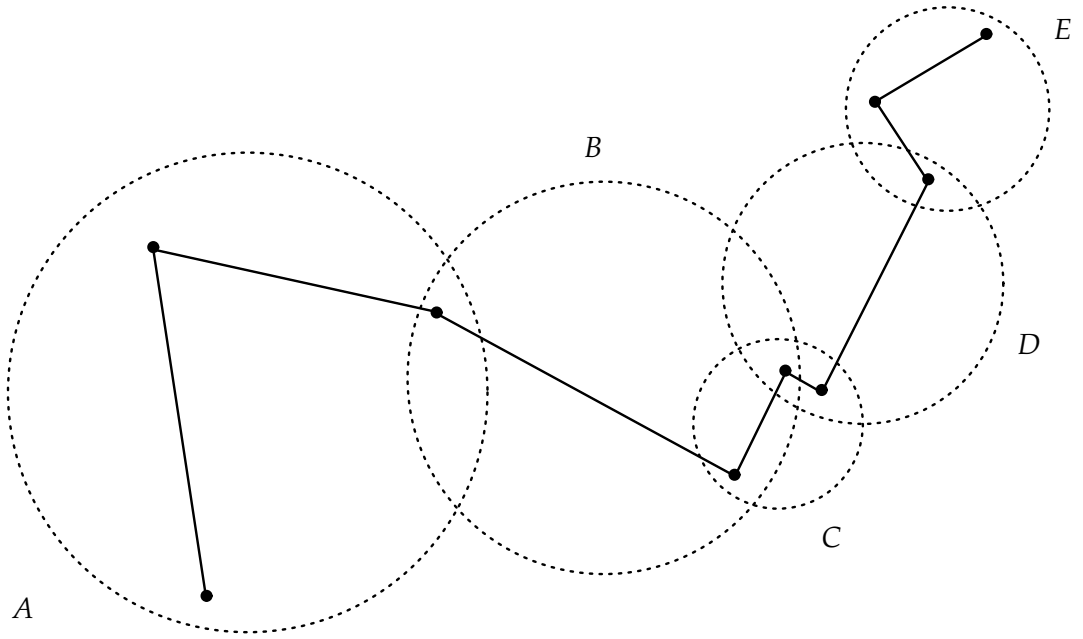


Figure 4.1. An example set of points for demonstrating the naïve heuristic. The solid lines are edges of the MST. The five circles each contain three points that can be used to give a reduction in length by finding their S-point.

One of the early heuristics is by Chang [5]. Chang's heuristic takes $O(n^4)$ time and for randomly generated problems gives reductions of about 3% on average. The method of finding sets of three points is more complex than the heuristic presented above. The three points can be either given points or already added S-points. The starting point is the MST and changes are made iteratively. At each step all possible changes are considered and the change giving the greatest reduction is accepted.

Smith *et al.* [37] give a very fast heuristic. The heuristic uses two geometrical constructions called the *Delaunay triangulation* and *Voronoi polygon* to find three and four point subsets which are connected by two and three edges of the MST respectively. Further, the edges must be part of triangles in the triangulation for which a S-point exists, and therefore a reduction is possible. Full Steiner trees are constructed for the subsets and replace the MST edges. The triangles are considered in decreasing order of reduction. The heuristic requires $O(n \log n)$ time and gives slightly better trees than Chang's heuristic. This heuristic, the Delaunay triangulation and Voronoi polygon are discussed in Section 4.2.

Beasley [3] does not use the triangulation or polygon but does use the same idea of considering three and four point connected subgraphs of the MST. S-points of the three or four point full Steiner trees giving reductions are added

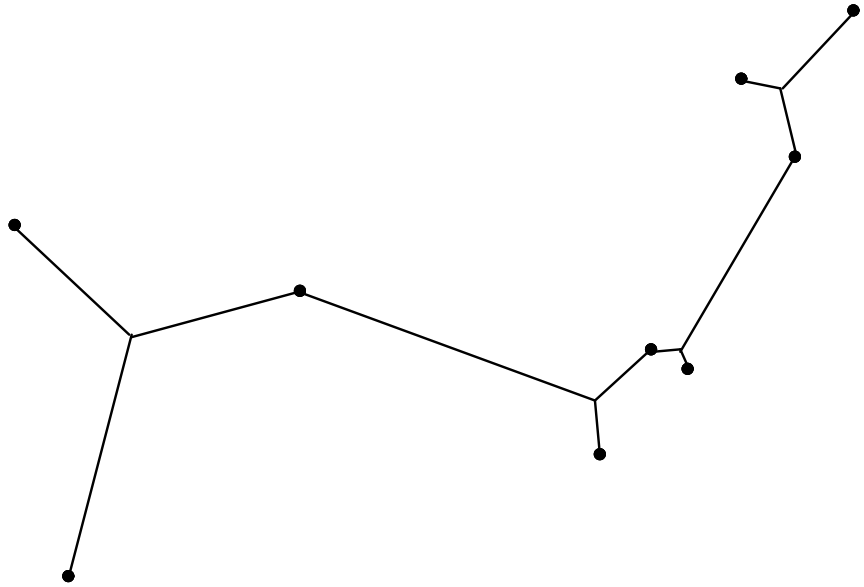


Figure 4.2. The approximate Steiner minimal tree found using the naïve heuristic.

to the set of points. The MST of the enlarged set of points is found and any added points are placed in their optimal position or removed depending on whether they have degree three or two respectively in the MST. The process is repeated until no further improvements are possible. Beasley's experimental results give a solution time of $O(n^{1.317})$ and better reductions than the Smith *et al.* heuristic. However, this is not surprising because Beasley considers all three and four point connected subgraphs of the MST, while Smith *et al.* investigate only a subset of these subgraphs.

Beasley and Goffinet [2] bring together the power of the Delaunay triangulation and the approach of Beasley's first heuristic to give a high quality method with an empirically based time requirement of $O(n^{2.19})$. The method is enclosed within a simulated annealing framework to overcome cycling problems that occur from time to time. This heuristic is discussed in Section 4.3.[2]

A heuristic by Lundy [27] was developed for finding evolutionary trees. Simulated annealing based on randomly moving from one full Steiner topology to another is used to find a full Steiner tree for a set of points. Lundy found this method compared favourably with existing methods for finding evolutionary trees.

Hesser *et al.* [16] implemented a genetic algorithm to solve the Euclidean Steiner tree problem. A simulated annealing algorithm approach was also developed for comparison purposes. Both provide good solutions in reasonable times. In both algorithms the positions of S-points are randomly perturbed, and a minimum spanning tree is found for the given points and S-points. Local optimisation of the MST is also performed: S-points of degree one and the

[2]Simulated annealing is introduced in Chapter 5.

edges incident on such points are removed, and for S-points of degree two the points and incident edges are removed and the adjacent points are reconnected by direct links.[3]

4.2 Smith, Lee and Liebman's Heuristic

One of the better heuristics for the Euclidean Steiner tree problem is by Smith *et al.* [37]. It is based on adding a S-point to a set of three points which form a triangle that has two edges in common with the minimum spanning tree. The heuristic uses two geometrical structures called the *Voronoi polygon* and *Delaunay triangulation* to find triangles and pairs of triangles which may give a reduction in length of connection if S-points are added.[4]

4.2.1 Voronoi polygons and Delaunay triangulations

The Voronoi polygon and Delaunay triangulation are fundamental structures in computation geometry.[5] For a set a points $\mathcal{A} = \{a_1, \ldots, a_n\}$ in two dimensional Euclidean space the Voronoi polygon is a collection of regions $\{V_1, \ldots, V_n\}$ such that any point in V_i is closer to a_i than to another point in $\mathcal{A}$. Figure 4.3 shows the Voronoi polygon for a set of fifteen points. The Delaunay triangulation is the unique triangulation of $\mathcal{A}$ such that no circumscribing circle of any triangle contains any points in $\mathcal{A}$. Figure 4.4 shows the Delaunay triangulation of the same set of fifteen points.

The Voronoi polygon and Delaunay triangulation are intimately related. They form a dual pair, that is given either it is possible to find the other. The Voronoi polygon is made up of perpendicular bisectors of segments joining points in $\mathcal{A}$. The intersections of the bisectors are called *Voronoi points*. Each Voronoi point is the center of a circumscribing circle of a triangle in the Delaunay triangulation. This is shown in Figure 4.5.

The construction of either a Voronoi polygon or Delaunay triangulation is a non-trivial task but is not exceptionally complicated. The problem with practical algorithms is coping with the inherent inaccuracy of using the floating point arithmetic of computers. For example, if the three points a_1, a_2 and

[3]Genetic algorithms are a random search technique for finding global optimum. They are based on an analogy with natural evolution.

[4]Professor James MacGregor Smith (`JMSMITH@ecs.umass.edu`) of the University of Massachusetts kindly made his FORTRAN implementation of the heuristic available. The maximum number of points it can handle was eighty, this was increased to two hundred. Unfortunately it appears the Delaunay triangulation component of the implementation is unreliable for large point sets (in excess of fifty or sixty) especially when points are very close to one and other. Scaling of the coordinates did not alleviate this problem. However, use of the program and assistance offered by Professor Smith were invaluable.

[5]Fortune [12] is a survey of the properties and practical algorithms for constructing the two structures. It isn't confined to just two dimensions but instead discusses the polygon and triangulation in d-dimensional Euclidean space. The Voronoi polygon's primary use is in answering *what is the nearest point?* questions. It has been used in crystalography, the study of equilibrium states of alloys, the prediction of rainfall and pattern recognition.

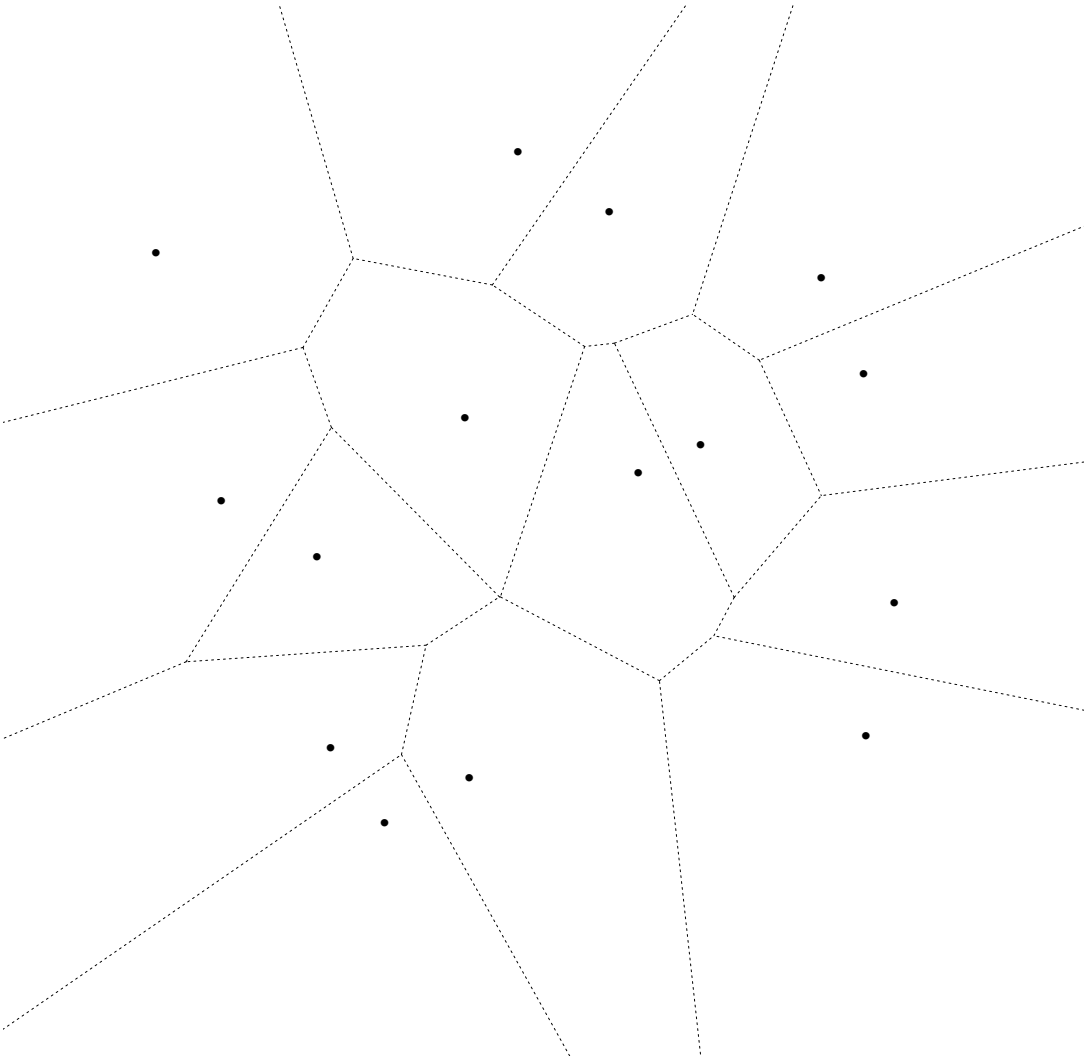


Figure 4.3. The Voronoi polygon for fifteen points. For any point in a region the closest *given point* is the given point within the same region.

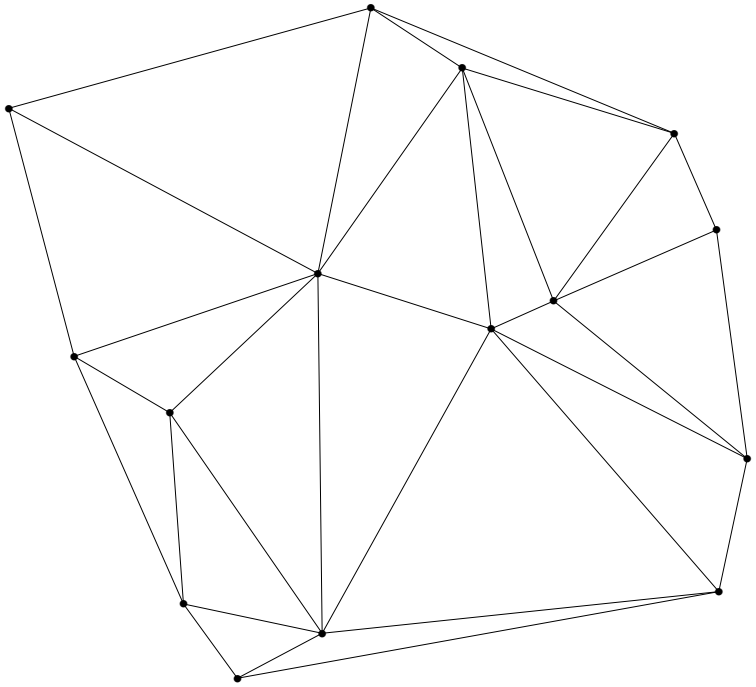


Figure 4.4. The Delaunay triangulation for fifteen points. The circumscribing circle of each triangle contains no given points.

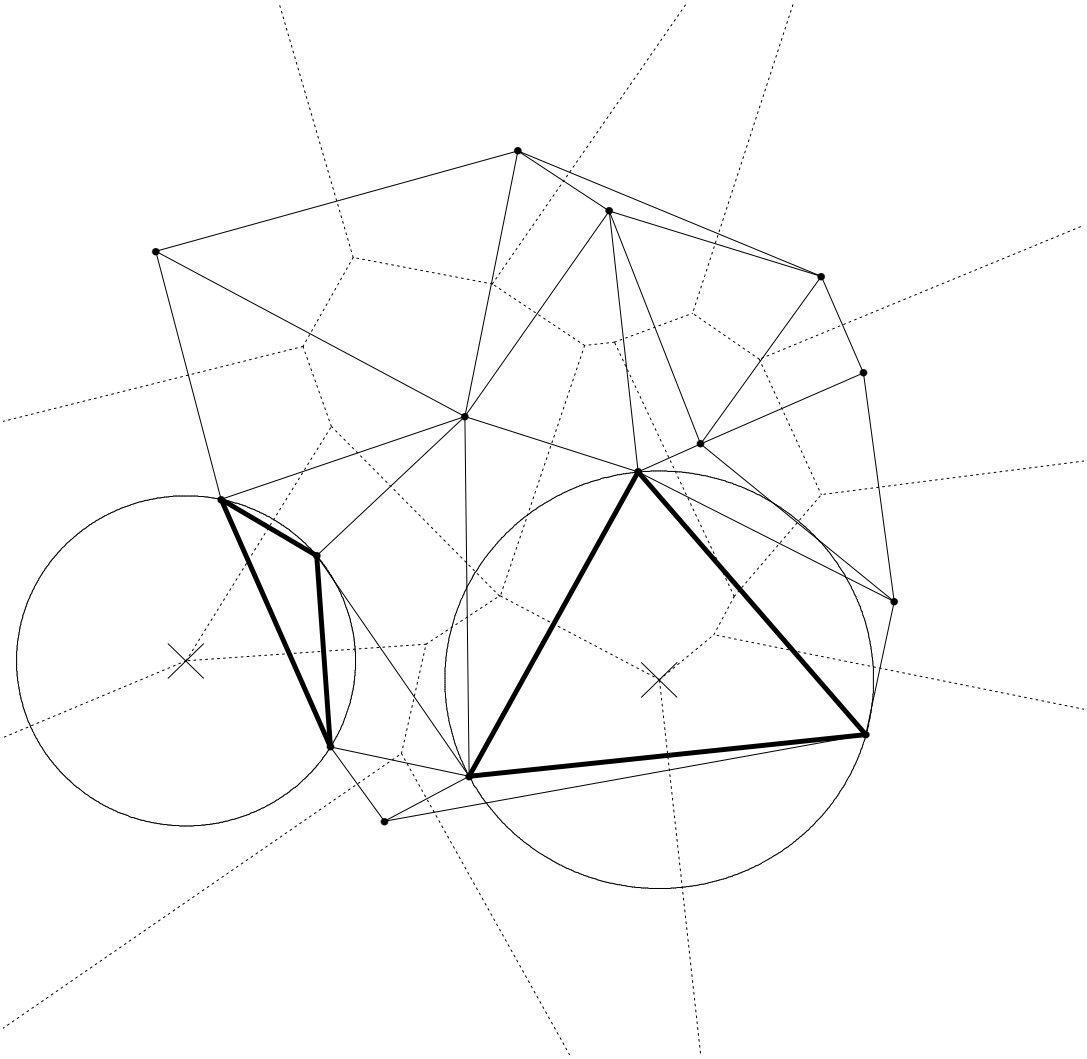


Figure 4.5. The Voronoi polygon and Delaunay triangulation of the fifteen points. The circles are the circumscribing circles of two of the triangles. The centre of each circle is the corresponding Voronoi point of the triangle (shown as crosses).

a_3 form a right angle at a_2 and a fourth point a_4 is introduced very close to a_2 then it is possible a_1, a_2, a_4 will be computed as being colinear and similarly for a_2, a_3, a_4. This leads to the conclusion that a_1, a_2, a_3 are colinear! Some implementations use exact integer arithmetic and suitable scaling of coordinates, however this is at the expense of increased computation time. Most algorithms find the Delaunay triangulation and then the Voronoi polygon from the triangulation. To go in the reverse direction is dangerous as the Voronoi polygon is defined by computed Voronoi points. Some freely available floating point implementations are:

- `voronoi` by S. Fortune and available by anonymous `ftp` from `netlib.ornl.gov`;
- `qhull` by The National Science and Technology Research Center for Computation and Visualization of Geometric Structures at the University of Minnesota and available by anonymous `ftp` from `geom.umn.edu`;
- `DELAUNAY_TREE` by O. Devillers is a C++ object within the LEDA library [30] and available by anonymous `ftp` from `ftp.th-darmstadt.de` or `ftp.uni-sb.de`.

The properties of Delaunay triangulations of particular interest in finding solutions to the Euclidean Steiner tree problem are:

- The minimum spanning tree is a subgraph of the Delaunay triangulation;
- Of all possible triangulations the Delaunay triangulation maximises the smallest angle in any triangle;
- A Delaunay triangulation in two dimensions can be found in $O(n \log n)$ time;[6]
- There are $O(n)$ triangles in a Delaunay triangulation.

Together these properties mean the Delaunay triangulation provides a way of quickly identifying triangles which can be used to reduce the length of the minimum spanning tree. In particular, the second property means there are triangles similar to equilateral triangles from which the greatest reduction in length can be achieved. How the Delaunay triangulation (and Voronoi polygon) are actually used is described below.

4.2.2 The heuristic

Smith *et al.* [37] use a two phase approach for finding an approximate SMT, denoted by $\widetilde{\text{SMT}}$. The two phases are called *reduction* and *expansion*. The first involves finding triangles and pairs of triangles in the Delaunay triangulation which have edges in common with the MST. The Voronoi polygon is used to

[6]The time grows exponentially with dimension. Most algorithms are of practical use for dimensions of less than five or six.

find "best" pairs. To these triangles, or pairs of triangles, are added one or two S-points, if possible, to give a shorter connection of the points of the triangles. The second phase involves the concatenation of the locally optimal connections of points to give $\widehat{\text{SMT}}$. The heuristic is described by solving a simple ten point example. The coordinates of the points are shown in Table 4.1. The set of points has a non-degenerate Steiner polygon.

1	(8,31)	6	(72,64)
2	(23,63)	7	(74,21)
3	(31,72)	8	(87,56)
4	(48,48)	9	(88,110)
5	(49,8)	10	(118,29)

Table 4.1. The example ten point problem coordinates.

Reduction

Figure 4.6 shows the minimum spanning tree, the Delaunay triangulation and Voronoi polygon of the set of points. Triangles can easily be identified that have two edges in common with the MST and for which a S-point exists. For example, the triangle defined by the three points 1, 2 and 4 has a S-point at coordinate (26.65,52.25) giving a Steiner ratio of 0.952 for the triangle. Adding this S-point, removing the two MST edges and replacing them with edges from the three points to the S-point gives a shorter connection. This is the heart of the heuristic, triangles and S-points are considered in a systematic manner, giving a shorter connection, an approximate SMT.

Table 4.2 shows S-points and Steiner ratios of those triangles with two edges in common with the MST and for which a S-point exists. Further, the triangles are shown in order of increasing Steiner ratio. This is the order in which they are considered in the expansion phase. These triangles form the set of *candidate triangles* from which reductions in the MST are sought.

Triangle	Steiner Point	Steiner Ratio
(7,8,10)	(91.34,37.16)	0.904
(2,3,4)	(29.45,65.03)	0.949
(1,2,4)	(26.65,52.25)	0.952
(6,7,8)	(82.01,54.52)	0.984
(6,8,9)	(77.10,65.48)	0.987

Table 4.2. The example problem candidate triangles.

Expansion

The expansion phase is a process of building $\widehat{\text{SMT}}$ by concatenating candidate triangles. However, it is not simply a case of combining these triangles

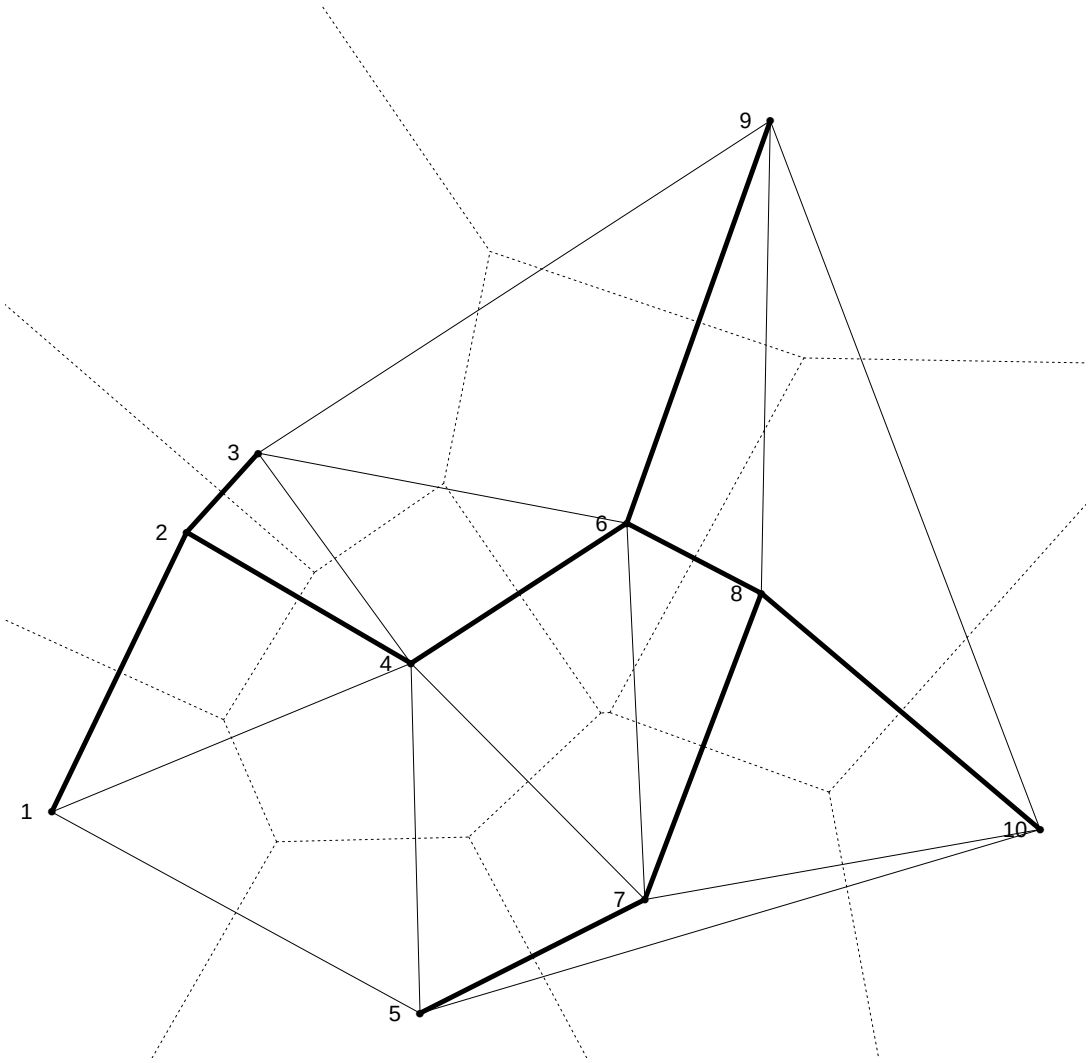


Figure 4.6. The minimum spanning tree (bold line), Delaunay triangulation and Voronoi polygon of the ten points.

without creating cycles, instead pairs of triangles, with at least one from the candidate set, are considered. For suitable pairs four point full Steiner trees are constructed. The advantage of this over concatenating individual triangles is that often a four point Steiner tree gives a better reduction. This is the experience of Smith *et al.* They go further and consider forming full Steiner trees of points making up k triangles. But results show that the additional effort is not rewarded by proportionate improving reductions.

Smith *et al.* devise a simple rule for determining which triangles should be considered for pairing. The rule uses the Voronoi polygon to find the *"most regular" convex quadrilateral* formed with neighbouring triangles.[7] Only triangles that give a reduction on the MST are used, that is their Steiner ratio is less than one. It is not necessary for a neighbour triangle to be in the candidate set for it to be considered for pairing. For a triangle t_i with corresponding Voronoi point v_i the most regular convex quadrilateral formed with neighbouring triangle t_j is with the neighbour that minimises the distance from v_i to v_j. Figure 4.7 shows two most regular convex quadrilaterals.

A square doesn't give the maximum possible reduction, but experience shows that very good reductions are possible using the most regular rule. Convexity is a necessary requirement for existence of a FST, and helps give good reductions.[8] And empirical results indicate that increasing the number of S-points tends to increase the reduction. This is generally true for convex sets of points with no interior points. Using the Delaunay triangulation and Voronoi polygon for constructing the four point sets guarantees there are no interior points.

Only certain pairs of triangles will give a reduction over the MST. The triangles must be neighbours and must be connected by three edges of the MST. A reduction in the connection of the four points using a four point FST will necessarily reduce the length of the overall MST. Further, if the FST is to be concatenated to the growing $\widehat{\text{SMT}}$ then the points making up the triangles must belong to disjoint components of $\widehat{\text{SMT}}$, otherwise the concatenation of the four point FST to the growing $\widehat{\text{SMT}}$ will create a cycle. These required properties of pairs are shown by example below.

The concatenation process begins with an empty $\widehat{\text{SMT}}$. The triangles forming the list of ordered candidate triangles are considered in turn. For each triangle an attempt is made to form a pair. If a suitable pairing exists, the four point FST, if it exists, is added to $\widehat{\text{SMT}}$, otherwise the three point FST of the candidate triangle is added to $\widehat{\text{SMT}}$. In both cases cycles must not be created. When all candidate triangles have been processed, it may be necessary to connect disjoint components of $\widehat{\text{SMT}}$ and individual points not in any component. This is done using MST edges.

[7] "Regular convex quadrilateral" is a long winded way of saying "square".

[8] For the FST to exist it is also necessary that the two E-circles must not intersect and the axis must intersect both E-arcs.

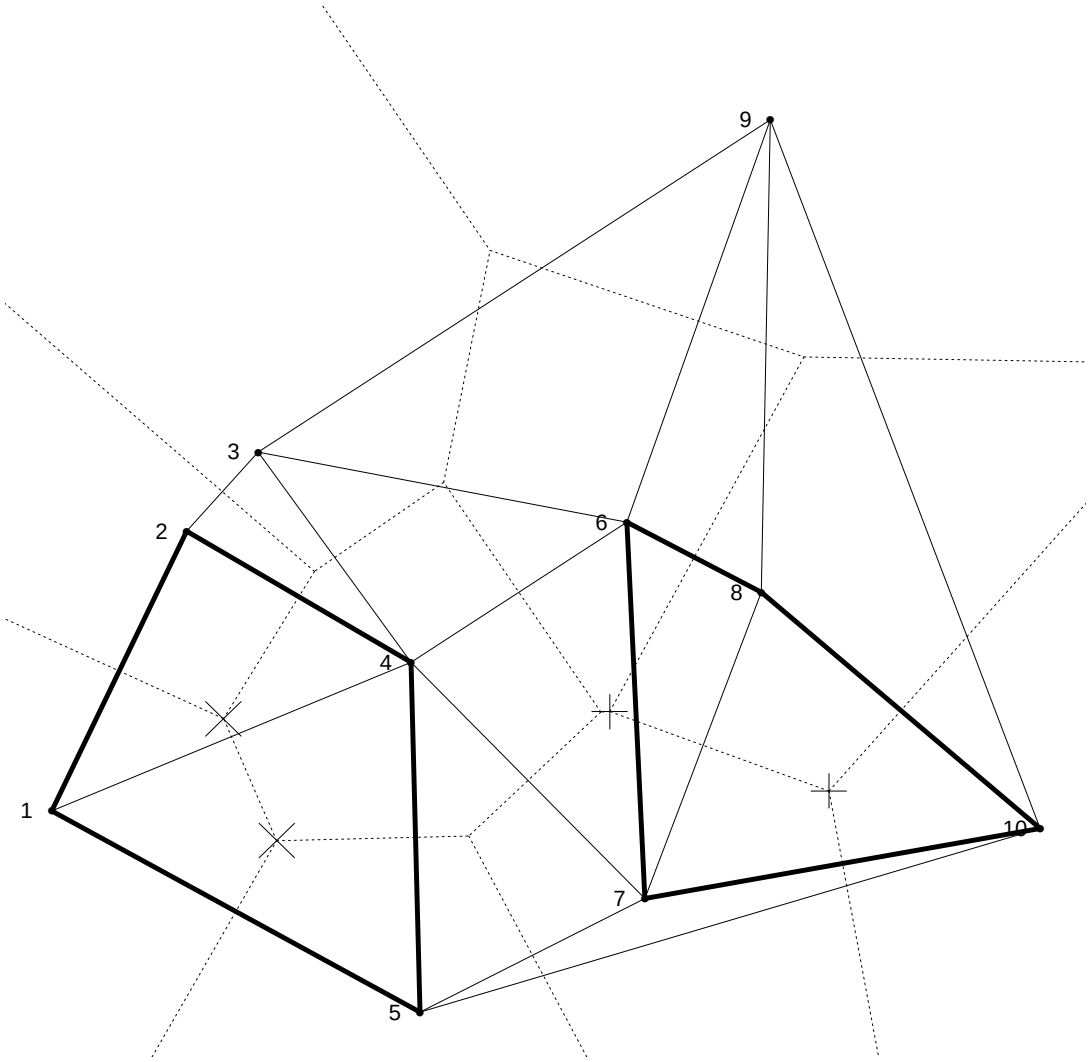


Figure 4.7. Two most regular convex quadrilaterals. The left hand quadrilateral is formed by combining triangle (1,4,5) with its "nearest" neighbour triangle (1,2,4). The crosses show the Voronoi points of the two triangles. Similarly on the right, triangle (7,8,10) has "nearest" neighbour triangle (6,7,8).

The example problem's $\widehat{\text{SMT}}$ is constructed by the following sequence of parings and concatenations:

1. The first candidate triangle is (7,8,10). It forms a most regular convex quadrilateral with neighbour (6,7,8) (see Figure 4.7). The four points are connected by three edges of the MST (see Figure 4.5). Unfortunately neither four point FSTs exist for the points. Therefore, the three points 7, 8 and 10, and their S-point (91.34,37.16) are added to $\widehat{\text{SMT}}$.

2. The next candidate triangle is (2,3,4). It can be paired with (3,6,4). One of the FSTs does exist (and in fact is the SMT for the four points). The two S-points are (30.44,66.37) and (48.53,53.38). Points 2 and 3 are connected to the same S-point. Concatenating this FST to $\widehat{\text{SMT}}$ does not create any cycles. $\widehat{\text{SMT}}$ now comprises two components: one connecting points 2, 3, 4, and 6, and the other connecting 7, 8, and 10.

3. Candidate triangle (1,2,4) isn't considered because a cycle is created if added to $\widehat{\text{SMT}}$.

4. Similarly for triangle (6,7,8).

5. The final candidate triangle is (6,8,9). The three point FST of the triangle with S-point (77.10,65.48) is concatenated with $\widehat{\text{SMT}}$ without creating a cycle. This addition now means $\widehat{\text{SMT}}$ is one component connecting all but 1 and 5.

6. Points 1 and 5 are connected to $\widehat{\text{SMT}}$ by MST edges.

The approximate SMT is shown in Figure 4.8. The MST length is 277.71, the length of $\widehat{\text{SMT}}$ is 266.55. The heuristic gives a reduction 4.02%. The optimal SMT (also shown in Figure 4.8) has a length of 266.23, a reduction of 4.13%.

4.2.3 Possible changes to the heuristic

The heuristic is fast and gives good reductions using simple rules for ordering triangles and pairing triangles. However there are a range of changes to both the rules and method of processing the list of triangles that can possibly improve the reductions, but generally at the expense of longer computation times. As with many heuristic techniques there are advantages and disadvantages and experimenting is often the only means of investigating performance.

The order of processing candidate triangles

Sarkar [33] orders the candidate triangles by decreasing reduction in length instead of increasing Steiner ratio. Smith [38] considered many different orders but strangely enough not that suggested by Sarkar. However, Smith's experience was that the ratio worked best of the methods of ordering he considered. A justification for using length is: length is the property being minimised,

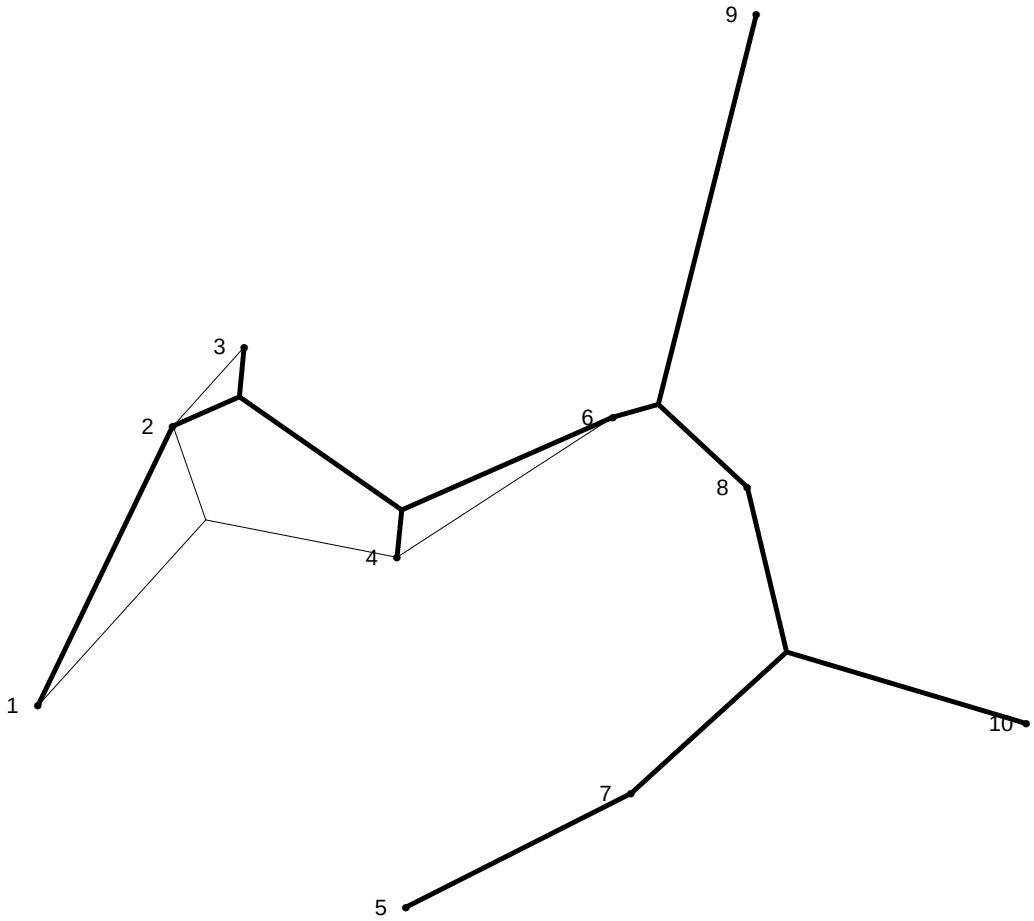


Figure 4.8. The example problem approximate SMT (bold line) and optimal SMT. In the optimal SMT there is a S-point very close to point 2.

therefore processing triangles ordered by reduction in length is a more direct approach than using triangles that have small Steiner ratios. Of course, there is likely to be a high correlation between reduction in length and Steiner ratio. It is possible using the ratio order that concatenating a relatively small triangle, but with a good ratio, may prevent a larger triangle, with a probable large reduction in length, from being concatenated in later processing. This is because a cycle will be created.

Using the example of the previous section, the list of candidate triangles ordered by reduction in length is slightly different to the ordered by ratio list. Triangles (1,2,4) and (2,3,4) swap places. The revised order is show in Table 4.3.

Triangle	Steiner Point	Reduction in Length
(7,8,10)	(91.34,37.16)	7.531
(1,2,4)	(26.65,52.25)	3.096
(2,3,4)	(29.45,65.03)	2.101
(6,7,8)	(82.01,54.52)	0.869
(6,8,9)	(77.10,65.48)	0.854

Table 4.3. The example problem candidate triangles ordered by reduction in length.

The processing of the list begins as before with triangle (7,8,10) able to be paired but not giving a four point FST. This triangle and its S-point become $\widehat{\text{SMT}}$. Triangle (1,2,4) forms a most regular convex quadrilateral with neighbour (1,4,5) but can not be paired because the four points are not connected by three edges of the MST. Therefore triangle (1,2,4) and its S-point are successfully concatenated to $\widehat{\text{SMT}}$. Neither of the next two candidate triangles can be concatenated because cycles would be created. The last triangle does not form any cycles and can not be paired, it is added to $\widehat{\text{SMT}}$. To complete $\widehat{\text{SMT}}$, MST edges from 4 to 6, and 5 to 7 are required. The approximate SMT obtained is very similar to the optimal SMT. To two decimal places its length is identical to the optimal SMT. However, it is not optimal because the angle made by the edges incident at point 2 is less than 120°. The difference between $\widehat{\text{SMT}}$ and the optimal SMT is the S-point (23.07, 62.99) which is very close to point 2 (23,63). In this instance using an alternative ordering has given a better $\widehat{\text{SMT}}$ than the standard heuristic.

Second and third most regular convex quadrilaterals

The heuristic attempts to form a most convex quadrilateral by pairing a candidate triangle with the neighbour triangle whose Voronoi point is closest to the candidate's. Further, the neighbour must be such that the four points of the paired triangles are connected by three edges of the MST. If neither the most regular convex quadrilateral is a valid pairing nor a four point FST of the quadrilateral exists, then the candidate triangle alone is concatenated, if possible.

An alternative is to consider the next most regular quadrilateral, and if this is also unsuccessful the third most regular, if they exist. Using the example above with the list ordered by reduction in length it would have been possible to consider the second most regular pairing of candidate triangle (1,2,4) with neighbour (2,3,4). In this situation the heuristic would find the optimal SMT.

More complicated processing procedures

Sarkar [33] uses the list in a more complicated fashion. Two ideas are introduced. The first is to process the list without forming $\widehat{\text{SMT}}$, and instead four point FSTs are inserted into the list. The possibly larger list is then used to build $\widehat{\text{SMT}}$. The second idea is to perform an exhaustive search of the list using the disjoint set union procedure with the addition of backtracking in the event of creating infeasible unions or unions exceeding the length of the best known $\widehat{\text{SMT}}$. Clearly, both ideas introduce more computation, the first to a lesser extent than the second.

4.3 Beasley and Goffinet's Heuristic

Beasley and Goffinet [2] present a heuristic for the Euclidean Steiner tree problem that uses the Delaunay triangulation iteratively to create more and more potential S-points. Whereas the heuristic described above uses the Delaunay triangulation only once to generate candidate triangles, in this heuristic it is used over and over again together with the minimum spanning tree on a set of points that changes as the heuristic progresses. Further, all triangles are considered at the same time and not in a sequence determined by some ordering, usually the Steiner ratio. Simple rules are used to eliminate S-points and give a connection that is a Steiner tree, although not necessarily the shortest Steiner tree. The heuristic is not as fast as that of Smith *et al.* but experience indicates it produces better quality solutions. The heuristic and computational experience are described below. An example is given in Appendix C.[9]

4.3.1 The basic heuristic

Beasley and Goffinet repeatedly apply three stages of computation, which together form an *iteration*. The stages are called *expansion*, *reduction* and *re-expansion*. Unfortunately when compared to Smith *et al.* the names of stages and what they do have been swapped. For this heuristic expansion involves creating S-points using the Delaunay triangulation, and reduction means using the MST to remove redundant S-points. Smith *et al.* use reduction for finding S-points and candidate triangles, and expansion for concatenating candidate triangles and quadrilaterals to form a feasible tree.

[9] Dr John Beasley (`j.beasley@ic.ac.uk`) of Imperial College in London kindly provided a preprint of the paper. The notation of Beasley and Goffinet [2] is used in this section.

The heuristic uses several objects, or variables, to control execution and keep track of best solution, current and previous solution.

- V is the set of given points;
- S is the current set of S-points, S_{last} is the previous set and S^* is the set of S-points giving the best solution found;
- L is the length of the MST of $V \cup S$, L_{last} is the length of the MST of $V \cup S_{last}$ and L^* is the length of the MST of $V \cup S^*$;
- N is the number of expansions allowed at the current iteration and N^* is the maximum number of expansions allowed.

Initially the sets of S-points are empty, $N = 1$ and the lengths are all equal to the length of the MST of V.

Expansion

This stage involves finding the Delaunay triangulation of $V \cup S$ N times. Each time and for each triangle its S-point is found, if it exists. The S-point is added to S. The number of S-points grows rapidly especially if N is greater than one.

Reduction

The MST of $V \cup S$ may possibly contain S-points that are not of degree three. The reduction stage eliminates these occurrences. Further, some S-points may not be in their optimal position, the points are moved.

1. The MST of $V \cup S$ is determined.
2. Any points in S which have degree less than or equal to two are removed from S, and execution returns to 1.
3. Any points in S of degree three which are not in their optimal position are moved to the optimal position. If there is no such optimal position then the point is removed from S. This step is repeated until no further reduction in length can be achieved by moving points.
4. Any points in S of degree four are removed from S and replaced by the S-points, if any, which optimally connect the four points to which the original S-point was connected.
5. Any points in S with degree five or more are removed.
6. If any changes occurred in any of steps 3, 4 and 5 then execution returns to 1.

Re-expansion

The last stage in an iteration is re-expansion. If there any edges in the MST of $V \cup S$ which share a common end point and make an angle of less than 120° at the point then a S-point is added to S. The S-point is based on the three points connected by the edges. If any S-points have been added in this stage then the heuristic returns to the reduction stage.

The end of an iteration

An iteration is completed when there are no changes in the re-expansion phase. The current solution is compared with the best and the best is updated if necessary, that is if $L < L^*$ then $L^* = L$ and $S^* = S$. If $L = L_{last}$ then $N = N + 1$, and if $N \leq N^*$ another iteration begins. However, if $N = N^*$ then the heuristic stops with a solution given by S^*. If $L \neq L_{last}$ then $N = 1$, $L_{last} = L$, $S_{last} = S$ and another iteration begins.

The effect of these rules is to repeat an iteration with more Delaunay triangulations in the expansion stage, and therefore more S-points being created, if the last iteration gave no change in solution. And if a change did occur another iteration is performed but with only one Delaunay triangulation being used to find more S-points. The $N = N^*$ test provides a stopping rule for the heuristic. Beasley and Goffinet use $N^* = 6$ in their experiments.

4.3.2 An important enhancement to the heuristic

Beasley and Goffinet's limited computational experience shows that the heuristic sometimes becomes "trapped". It cycles between two solutions and never stops. This is demonstrated in the example in Appendix C, although this is by chance rather than design of a suitable set of points. They overcome this nuisance by introducing a random component at the end of each iteration into the heuristic. As the number of iterations increases the probability of accepting an increasing length solution decreases. Eventually, a point is reached where the cycle is broken and the heuristic terminates.[10]

To overcome cycling variable T is used. This has an initial value of T_0. At the end of an iteration T becomes αT, where $0 < \alpha < 1$. The rules for deciding what to do next are amended. If $L < L_{last}$ then $N = 1$, $L_{last} = L$, $S_{last} = S$ and another iteration begins. If $L > L_{last}$ then the change in solution relative to the length of the MST of V, L_0, is used in a "coin tossing" experiment. The relative change is $C = \frac{L - L_{last}}{L_0}$. If a uniform random number is less than $e^{\frac{-C}{T}}$ then $N = 1$, $L_{last} = L$, $S_{last} = S$. Otherwise $N = N + 1$, $L = L_{last}$ and $S = S_{last}$.

[10]Beasley and Goffinet call this putting the heuristic within a simple simulated annealing framework. Simulated annealing is discussed in detail in Chapter 5. Whether to call it simulated annealing is debatable. Annealing uses many small random changes to a solution to find a optimal solution. This heuristic does not. Annealing starts from any solution. This heuristic always starts from the MST. *Regardless, the heuristic does work and produces very good solutions in reasonable times.*

The rules when $L = L_{last}$ and $N = N^*$ are not altered. Beasley and Goffinet use $T_0 = 0.7$ and $\alpha = 0.7$.

The effect of the above rules is to always accept a decreasing change in length, but to accept increasing length solutions with a probability that decreases as the heuristic progresses. If an increasing length solution is rejected the number of expansions is increased. This is where the cycle is broken. Eventually, the increasing cost is rejected N^* times and the heuristic stops.

4.3.3 Computational experience

Beasley and Goffinet's experience indicates the heuristic is the best found to date. They experiment using randomly generated problems and in particular Cockayne and Hewgill's one hundred point problems with known optimal solutions (see Appendix B). Analysis of the execution times gives an empirical relationship of $O(n^{2.19})$. An interesting feature of the heuristic was that the number of Delaunay triangulations required appears to be independent of the problem size. Problems on average needed about fifty Delaunay triangulation calculations. For Cockayne and Hewgill's problem the heuristic found the optimal solution in two of the thirty problems. The heuristic gave solutions that were on average 0.115% above the optimal length, with a worst performance of 0.470% above optimal. Beasley and Goffinet do not discuss the effects of changing the "annealing" parameters T_0 and α, or the maximum number of expansions parameter N^*. This parameter is six in their experiments.

Beasley and Goffinet's results for Cockayne and Hewgill's problems are used in Chapter 7 as the benchmark for comparing performance of simulated annealing with other heuristics for the Euclidean Steiner tree problem.

Chapter 5

Simulated Annealing

In Chapter 4 several heuristics for the Euclidean Steiner tree problem were presented. In this chapter a type of heuristic called *simulated annealing* is presented. It is not a heuristic for a particular problem but a style or method of solution that can be applied to many problems. This chapter lays the mathematical foundations of simulated annealing. Chapters 6 and 7 investigate the application of simulated annealing to the travelling salesman problem and the Euclidean Steiner tree problem respectively. Chapter 6 is also an analysis of the behaviour of a particular implementation of simulated annealing.

Section 5.1 introduces general properties of heuristics for combinatorial optimisation problems, and highlights the strengths of simulated annealing compared to traditional heuristics. The physical process of annealing and the analogy resulting in simulated annealing are presented in Section 5.2. A mathematical model of the simulated annealing algorithm and properties of the model are discussed in Section 5.3. Section 5.4 considers the requirements for implementing a practicable simulated annealing algorithm, and describes a particular implementation of the simulated annealing algorithm. Finally, modifications to the standard simulated annealing algorithm are discussed in Section 5.5.

5.1 Combinatorial Optimisation and Heuristics

Combinatorial optimisation is the process of finding the the *globally optimal configuration* of discrete variables with respect to some function of the variables. Many combinatorial optimisation problems are very difficult and are *NP-hard*.[1]

A large number of combinatorial problems are of practical interest and importance, examples are the travelling salesman problem, timetabling, routing and scheduling, and layout and placement problems. Not being able to determine the globally optimal solutions to such problems in realistic amounts of time has encouraged the study of *approximate algorithms*. An approximate algorithm or *heuristic* should be able to find a configuration that is "close" to

[1]Section 2.3 is an introduction to computational complexity, in particular NP-completeness and NP-hardness.

a global optimum and do so in a reasonable amount of time. Unfortunately many heuristics are designed with only the immediate problem in mind and are therefore of limited use when attempting to solve other combinatorial problems. Algorithms of this sort are called *tailored algorithms*.

A technique used in many heuristics is *iterative improvement* or *local search*. From a starting configuration the immediate *neighbourhood* of configurations is considered. A neighbour is selected in some way and the cost of this configuration is compared with the current configuration. If the neighbour has a lower cost the current configuration is replaced by the neighbour.[2] If not then other neighbours are considered. The process stops when all neighbours have been considered without any reduction in cost. The advantages of iterative improvement are:

- A single run of the process can be done very quickly on average;
- Given this, many runs can be made from different starting points;
- The process is generally applicable. Three main parts are required for the process: a means of describing a configuration, an objective or cost function and some way of describing a neighbourhood or of generating possible neighbours.

Unfortunately iterative improvement has its disadvantages:

- The process stops at the first *local* optimum found. It may not necessarily be the global optimum and there is no way to "back out" and look elsewhere;
- Where the process stops is dependent on where it starts. However this is overcome, in part, by trying different starting points;
- In most cases it is not possible to give an upper bound on the computation time.

Iterative improvement's downfall is its blinkered view of the world. It only accepts configuration changes that give a decrease in cost. *Simulated annealing* on the other hand not only accepts decreasing moves but tolerates increasing moves. Thus it is able to escape the curse of local optimum and look elsewhere for the elusive global optimum. Simulated annealing has the advantages of standard iterative improvement but is able to shake off some, if not all, of the disadvantages *most* of the time. The reason for using "most" is that simulated annealing has a probabilistic component and many of its important properties are asymptotic in nature and are seldom fully realised. Regardless simulated annealing is a generally applicable, high quality combinatorial optimisation tool. But is often unable to compete with tailored algorithms because of the sometimes excessive computation times.

[2] Assuming the problem is a minimisation problem.

5.2 Simulated Annealing — The Overview

Simulated annealing resulted from the observation of the analogy between the physical process of annealing and of finding a global optimum for a combinatorial optimisation problem (Kirkpatrick *et al.* [23] and Černy [4]). The physical process and the analogy are the subjects of the following two sections.

5.2.1 Statistical physics, annealing and the Metropolis algorithm

Statistical physics is the study of aggregate properties of many particle systems. For such systems only the most probable behaviour of the system is observed. This most probable behaviour is given by the average of the system and fluctuations about the average at a given temperature. This average is characterised by the *thermal equilibrium*. This equilibrium is not a static equilibrium but one in which the system randomly changes from state to state. The probability distribution of the states is the *Boltzmann distribution*. The probability of finding the system in state i is given by

$$C(T)\exp\left(-\frac{E_i}{k_B T}\right)$$

where E_i is the energy of the system when in state i, T is the temperature, k_B is Boltzmann's constant and $C(T)$ is a normalising function.

Of interest is the behaviour of the system at very low temperatures. *Is a crystalline structure formed or is a glass formed?*[3]

At low temperatures the lower energy states, or ground states, and states with energies close to the ground states dominate and as the temperature approaches zero only the minimum energy states have a non-zero probability of occurrence.

Experiments to observe ground states of substances use the technique of *annealing*. This is the process of melting a substance and then cooling it slowly and spending long periods of time at very low temperatures close to the freezing point so that thermal equilibrium is achieved. If this process is done carefully then the ground state or minimum energy state of the substance is found. However, if the cooling is too fast then a metastable or locally optimal state can result, this process is called *quenching* instead of annealing.

The annealing process was modeled by Metropolis *et al.* [29] and is known as the *Metropolis algorithm*. From a collection of atoms at a given temperature T, an atom is selected and given a small random displacement. The change in the system's energy is $\triangle E$. If $\triangle E < 0$ the change is accepted, if $\triangle E \geq 0$ then the change is accepted with probability $\exp\left(-\frac{\triangle E}{k_B T}\right)$, where k_B is Boltzmann's constant.[4] If a large number of changes at temperature T are considered then the system approaches thermal equilibrium at temperature T, and the probability distribution of the states is the Boltzmann distribution. At this point

[3] A glass is a solid with no structure and with possibly many defects.

[4] This criterion for accepting or rejecting a change in configuration is called the *Metropolis criterion*.

the temperature is lowered by a small amount and the process of attaining thermal equilibrium repeated. As the temperature decreases the probability of accepting an increase in energy decreases. Therefore when the temperature is very low only decreasing changes are accepted. At some point no further changes remain to be investigated and the process terminates. If the annealing has been carefully and slowly performed then the final state of the system is a globally minimal state, a state of lowest energy. The globally minimal states are the only states with a non-zero probability of occurrence.

5.2.2 The analogy with combinatorial optimisation

The analogy of physical annealing with combinatorial optimisation is that there are many states (the positions of the atoms — configurations of the optimisation problem), and a quantity is minimised (the energy of the atoms — the cost of the optimisation problem).

The analogy can be extended to the Metropolis algorithm. The configurations of the optimisation variables are equivalent to the positions of the atoms, the cost is the energy and a quantity called the *control parameter*, c, takes the place of temperature. Application of the Metropolis algorithm to a combinatorial optimisation problem takes the following course: starting at a high value of c with an initial configuration i, a series of trials are performed where another configuration j is generated, j being a neighbour of i, the costs of the two configurations are compared and if $\triangle C_{ij} = C(j) - C(i) < 0$ then the change is accepted, but if $\triangle C_{ij} \geq 0$ then the change is accepted with probability $\exp\left(-\frac{\triangle C_{ij}}{c}\right)$. A large number of trials are performed and therefore an "equilibrium" or a *stationary distribution* of configurations is obtained. The control parameter is lowered and the sequence of trials is repeated. When the control parameter has reached zero the process is terminated and the "frozen" configuration is the solution. The process just described is the "bare bones" of the simulated annealing algorithm. The algorithm is relatively straightforward and can easily be extended to new optimisation problems. Further, Laarhoven [25] provides some evidence that simulated annealing performs better than repeated iterative improvement if both are given the same amount of computation time.

Simulated annealing is sometimes described as a *randomised iterative improvement algorithm*. It uses the concepts of neighbourhood and of always taking decreasing cost moves, but with the possibility of making increasing cost moves in the hope of escaping from local optimum. It also has a *divide and conquer* aspect, it first deals with the large scale structure of a problem then concentrates on the fine detail. This is shown in Figure 5.1, the evolution of a travelling salesman problem solution is shown as a series of tours at different values of the control parameter. The problem is one with four distinct groups of cities to be visited. At high values of c the interim solutions are chaotic but as c decreases the number of links between groups decreases and the details of the tour within each group are worked on.[5]

[5]This is based on an example in Kirkpatrick *et al.* [23]

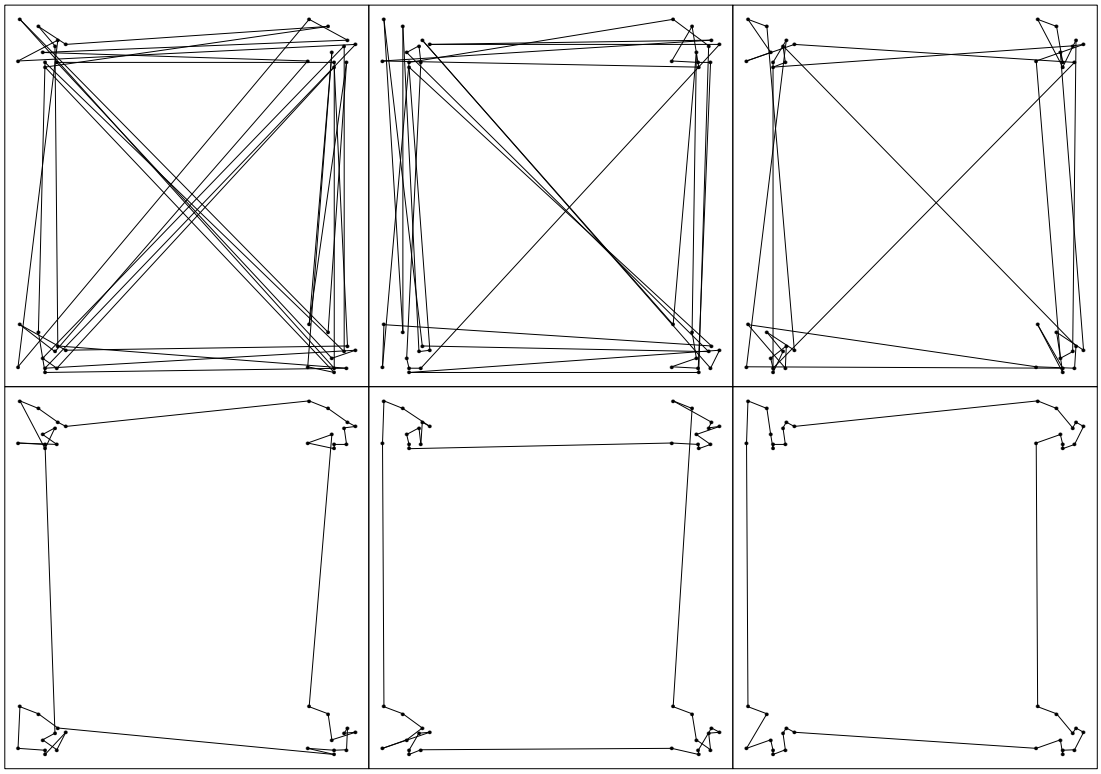


Figure 5.1. Evolution of a travelling salesman problem solution using simulated annealing.

To implement simulated annealing four components are needed:

- A description of a configuration;
- A generator of moves to other configurations or a neighbourhood description;
- An objective function;
- An *annealing schedule* or *cooling schedule*: a description of the sequence of control parameter values and number of trials at each value of c or a set of rules describing the calculation of these values.

5.3 Simulated Annealing — The Details

A mathematical model based on Markov chains is described in this section, and a set of conditions are presented which guarantee a simulated annealing algorithm finds a globally optimal configuration. The notation follows that of Laarhoven [25].

5.3.1 A mathematical model

The simulated annealing algorithm can be described using a Markov chain. Each trial of the annealing algorithm corresponds to a transition and the probability of a transition being accepted depends only on the costs of the current configuration and trial configuration. The probability of transition from configuration i to configuration j at the k^{th} trial where $X(k)$ is the outcome of the k^{th} trial is denoted by

$$P_{ij}(k) = P(X(k) = j \mid X(k-1) = i).$$

If this probability does not depend on k then the chain is called a *homogeneous Markov chain* otherwise it is an *inhomogeneous Markov chain*.

For simulated annealing the probabilities depend on the control parameter c, and if c is constant the transition probability is given by

$$P_{ij}(c) = \begin{cases} G_{ij}(c)A_{ij}(c) & i \neq j \\ 1 - \sum_{l \in R, l \neq i} G_{il}(c)A_{il}(c) & i = j \end{cases}$$

where R is the set of all configurations, G_{ij} is the probability of generating configuration j given current configuration i and A_{ij} is the conditional acceptance probability dependent on the costs of the configurations. Both probabilities are elements of the matrices $G(c)$ and $A(c)$ known as the *generation matrix* and *acceptance matrix* respectively. The matrix $P(c)$ with element P_{ij} is the *transition matrix*.

Two forms of the simulated annealing algorithm exist: the *homogeneous algorithm* and the *inhomogeneous algorithm*. The homogeneous algorithm is where

a number of transitions are generated at a fixed value of the control parameter. The control parameter is then decreased and the sequence of transitions is repeated. A decrement in the control parameter after every transition gives the inhomogeneous algorithm. The algorithms are described by either a sequence of homogeneous Markov chains or one inhomogeneous chain respectively.

5.3.2 Asymptotic convergence of the homogeneous algorithm

In this section conditions on $G(c)$ and $A(c)$ are described to guarantee the asymptotic convergence of the *homogeneous* algorithm to a globally minimal configuration.[6]

Convergence requires the existence of the *stationary distribution* of a homogeneous Markov chain. The stationary distribution is defined as as the vector q with the i^{th} component given by

$$q_i = \lim_{k \to \infty} P(X(k) = i \mid X(0) = j) \ \ \forall\, j.$$

If the stationary distribution exists then $\lim_{k\to\infty} P(X(k) = i) = q_i$ and the stationary distribution is the probability distribution of the configurations after an infinite number of transitions and is independent of the starting configuration.

Theorem 2.1 of Laarhoven [25] gives the following six conditions for the stationary distribution of configurations to converge to a uniform distribution on the set of globally minimal configurations as the control parameter is decreased to zero:

1. At any value of the control parameter c it must be possible to move from any configuration to any other configuration in a finite number of transitions;

2. All generation matrices $G(c)$ must be symmetric;

3. The probability of accepting a non-decreasing cost transition from configuration i to configuration k must be the same as the the probability of accepting the two step transition from i to k via configuration j where the cost of j is bounded below by the cost of configuration i and above by the cost of k;

4. The probability of accepting a decreasing transition is always one;

5. The probability of accepting an increasing transition is non-zero;

6. The probability of accepting increasing transitions must approach zero as the control parameter approaches zero.

[6] Similar conditions for the inhomogeneous algorithm can be found in Laarhoven [25] and Laarhoven and Aarts [24].

Two conditions can be relaxed without losing the guarantee of convergence:

- The first condition can be changed to:

  > At any value of the control parameter c it must be possible to move from any configuration to *a globally minimal configuration* in a finite number of transitions.

- The second condition can be changed to:

  > *Either* all generation matrices $G(c)$ must be symmetric *or are given by the uniform distribution over the neighbourhoods of each configuration, that is*
  >
  > $$G_{ij}(c) = \begin{cases} \mid R_i \mid^{-1} & j \in R_i \\ 0 & \text{otherwise} \end{cases}$$
  >
  > *where R_i is the set of neighbours of configuration i.*

Although this set of conditions provides a guarantee of finding a globally minimal configuration it requires each Markov chain to be of infinite length, and for there to be an infinite sequence of Markov chains. This is obviously unrealistic. However it is possible to approximate the asymptotic behaviour of the homogeneous simulated annealing algorithm arbitrarily closely in an exponential number of transitions. But this too may not be acceptable for large problems or when a solution is sought that is very close to a globally minimal solution. In Section 5.4.1 a method is described for approximating the asymptotic behaviour in polynomially bounded time but with the loss of the guarantee of finding the optimal configuration.

5.4 Cooling Schedules

A cooling schedule is a description of the values of the control parameter and number of transitions performed at each value of control parameter by a simulated annealing algorithm. The four components of a schedule are:

- A sequence of *finite* numbers of transitions to be performed at each value of the control parameter;
- A *finite* length sequence of control parameter values $\{c_k\}$ given by:
  - An initial value;
  - A rule for changing the value;
  - A final value.

The arguments for choosing one schedule over another or for designing a schedule are based on the underlying mathematical model of simulated annealing.

The general arguments for constructing a cooling schedule are:

- The initial value of c should allow virtually all transitions and the approximated stationary distribution of configurations at this value should be a uniform distribution over all configurations;
- The simulated annealing should stop when there has been no decrease in cost over a number of consecutive chains, that is the cost has not changed over a sequence of c_k;
- The number of transitions L_k and the change or *decrement* in c_k should be related to the idea of being close to the stationary distribution of configurations. The greater the change from c_k to c_{k+1} the longer it will take to regain the stationary distribution at c_{k+1}. Also, as c_k approaches zero transitions are accepted with decreasing probability so L_k approaches infinity.

5.4.1 A polynomial cooling schedule

A polynomial time cooling schedule is concerned with reaching a *quasi-equilibrium* in a time with an upper bound of some polynomial function of the problem size. A finite number of transitions L_k must be performed at control parameter value c_k with the objective of being "close" to the stationary distribution of the Markov chain for c_k, this is the quasi-equilibrium.

The schedule described in the next four sections is due to Laarhoven and Aarts [24] and Laarhoven [25]. It is a polynomial time schedule and therefore provides no guarantee of finding a global minimum. But it is relatively straightforward and although not giving the fastest execution times it requires little fine tuning to obtain good performance.[7]

The four components are the initial value of the control parameter, the decrement rule, the final value and the number of transitions at each value of the the control parameter. The schedule is characterised by three parameters: the *initial acceptance ratio*, the *distance parameter*, and the *stopping parameter*.

Each parameter corresponds to the first three components respectively in what it determines or controls, and these parameters are independent of the problem being solved. Whereas, the fourth component, the number of transitions L_k, is problem dependent.

Initial value of the control parameter

The initial value of the control parameter is determined by the *initial acceptance ratio* ϕ_0. It is found by performing a preliminary run of the algorithm starting with an arbitrary value for the control parameter. Statistics are gathered about the number of decreasing transitions, proposed increasing transitions and the increase in cost of such transitions.

[7] Laarhoven [25] gives empirical evidence based on runs over different instances of three different combinatorial optimisation problems. The polynomial cooling schedule is compared with two other schedules.

Suppose m_0 is the number of transitions to be attempted in the preliminary run, and during this run the number of decreasing cost and proposed increasing cost transitions are updated, m_1 and m_2 respectively, and the average proposed increase in cost $\overline{\triangle C}^{(+)}$ is also monitored. The approximate acceptance ratio is given by

$$\phi \approx \frac{m_1 + m_2 \exp\left(-\frac{\overline{\triangle C}^{(+)}}{c}\right)}{m_1 + m_2}.$$

This is rearranged to give the following expression for c with ϕ replaced by ϕ_0

$$c = \frac{\overline{\triangle C}^{(+)}}{\ln\left(\frac{m_2}{m_2\phi_0 - m_1(1-\phi_0)}\right)}. \tag{5.1}$$

The sequence of operations to obtain the initial value of the control parameter is:

1. The control parameter is given an arbitrary starting value;
2. A transition is attempted and m_1, m_2 and $\overline{\triangle C}^{(+)}$ are updated;
3. Equation 5.1 is used to find a new value of c;
4. Steps 2 and 3 are repeated until m_0 transitions have been attempted.

The last value of c is the initial value of the control parameter. The value of m_0 is arbitrary as the sequence of control parameter values reaches a stable value reasonably quickly. The value of ϕ_0 should be high, say around 0.9 or 0.95. This means most increasing cost transitions will be accepted and a uniform quasi-equilibrium can be obtained over all configurations.

Decrement rule for the control parameter

The decrease from c_k to c_{k+1} can either be large or small. Each option has its advantages and disadvantages. Small decreases mean quasi-equilibrium can be re-established faster (assuming a quasi-equilibrium existed at the end of the c_k sequence of transitions) but more sequences will be required. A large decrease results in fewer sequences but a much greater time to restore the quasi-equilibrium. The former approach of small decrements and shorter chains is used by Laarhoven [25] and is used here.

The stationary distributions of consecutive chains are considered close if for all configurations and values of k the following holds

$$\frac{1}{1+\delta} < \frac{q_i(c_k)}{q_i(c_{k+1})} < 1 + \delta \tag{5.2}$$

where δ is the *distance parameter* and $q_i(c)$ is the probability of being in configuration i in the stationary distribution for the chain corresponding to control parameter value c.

Theorem 3.1 of Laarhoven [25] gives a condition for Equation 5.2 to be satisfied. The condition can be rewritten to give an inequality expression for c_{k+1} in terms of c_k, δ and the difference in cost between a configuration i and a globally minimal configuration i^* denoted by $\triangle C_{ii^*}$. The condition is

$$c_{k+1} > \frac{c_k}{1 + \frac{c_k \ln(1+\delta)}{\triangle C_{ii^*}}} \quad \forall i, \quad k = 1, 2, \ldots \tag{5.3}$$

This is simplified by changing the set of all configurations to the set of "most probable" configurations occurring in the k^{th} Markov chain. This set is denoted by R_k and is defined by

$$R_k = \{i \in R \mid \triangle C_{ii^*} \leq \mu_k - C_{i^*} + 3\sigma_k\}$$

where μ_k and σ_k are respectively the mean and standard deviation of the costs of configurations in the k^{th} Markov chain.[8] Equation 5.3 is replaced by

$$c_{k+1} > \frac{c_k}{1 + \frac{c_k \ln(1+\delta)}{\mu_k - C_{i^*} + 3\sigma_k}}. \tag{5.4}$$

Obviously the optimal cost C_{i^*} is generally not known. Allowing for this, Equation 5.4 is written as an equality to give the decrement rule[9]

$$c_{k+1} = \frac{c_k}{1 + \frac{c_k \ln(1+\delta)}{3\sigma_k}}. \tag{5.5}$$

Equation 5.5 is the decrement rule for the control parameter of the polynomial cooling schedule. The sequence $\{c_k\}$ is not known at the beginning of a run of the simulated annealing algorithm. Each c_k is dependent on the previous value and on the standard deviation of the costs of the configurations generated in the previous Markov chain, as well as the distance parameter.

Final value of the control parameter

The final value of the control parameter is determined as the algorithm runs. It is not known at the start. It is not so much the value of c_k that determines whether the algorithm terminates but the level of the average cost of configurations generated in the k^{th} Markov chain relative to the average cost in the first chain as a function of the control parameter that precipitates termination.

As the algorithm runs the growing sequence $\{\mu_k\}$ is used to generate another sequence $\{\overline{\mu}_k\}$, where $\overline{\mu}_k$ is the smoothed value of μ_k. This smoothed sequence can be viewed as a function of the control parameter c and is used to define a function $\triangle\overline{\mu}(c) = \overline{\mu}(c) - C_{i^*}$ (although C_{i^*} is generally not known). If it is

[8] It is assumed the distribution of the costs can be approximated by the normal distribution. This is verified in Chapter 6 where the behaviour of the polynomial cooling schedule is investigated when applied to the travelling salesman problem.

[9] Laarhoven [25] states that using a small δ takes account of ignoring $\mu_k - C_{i^*}$ because $\frac{\mu_k - C_{i^*}}{\mu_1 - C_{i^*}}$ and $\frac{\sigma_k}{\sigma_1}$ as functions of c_k are practically identical. This is confirmed in Chapter 6.

accepted that a reasonable rule for terminating the algorithm is "$\triangle\overline{\mu}(c)$ *is small compared to* μ_1" and for $c \ll 1$

$$\triangle\overline{\mu}(c) \approx c\frac{\partial\overline{\mu}(c)}{\partial c}$$

then the algorithm is terminated if

$$\left|\frac{c_k}{\mu_1}\frac{\partial\overline{\mu}(c)}{\partial c}\right| < \epsilon_s \tag{5.6}$$

where the partial derivative is evaluated at c_k and is estimated from the sequence $\{\overline{\mu}_k\}$ and ϵ_s is the *stopping parameter*.

Equation 5.6 is the stopping condition for the polynomial time cooling schedule. It depends on the stopping parameter and the sequence of average costs of each series of generated configurations as a function of the control parameter.

Number of transitions

This is by far the simplest component of the polynomial cooling schedule to define but not necessarily the easiest to determine. The number of transitions at each value of the control parameter should be sufficient for the distribution of configurations to approach the stationary distribution. The number of transitions L_k is defined to be the size of the largest neighbourhood, that is

$$L_k = \max_{i \in R} \mid R_i \mid \quad k = 1, 2, \ldots \tag{5.7}$$

Laarhoven [25] compares the polynomial cooling schedule with other schedules. The empirical analysis applies the simulated annealing algorithm with the different schedules to the graph partitioning and travelling salesman problems. The results indicate that the polynomial schedule's solutions are often of inferior quality compared to another schedule's solutions. However, the difference is slight and is in part compensated for by the minimal tuning required of the polynomial schedule parameters. The distance parameter δ of the polynomial schedule determines the quality of the solutions, whereas the "better" schedule requires experimentation with three parameters. *In general, the quality of solutions is influenced little by the schedule used provided the reduction in cooling parameter is carried out carefully and accurately.*

5.5 Modifications to the Standard Simulated Annealing Algorithm

The simulated annealing described above relies on the analogy with the physical annealing process. Many changes have been suggested that move away from the analogy, however the algorithms are useful in solving real problems.

Eglese [11] gives an overview of simulated annealing and includes a survey of modifications to the standard algorithm.[10] Some of the modifications are easily implemented, others are more complicated, for instance a parallel version of simulated annealing. Four possible modifications are:

Keeping the best solution. This modification almost goes without saying. To keep track of the best configuration and its cost has little effect on the running time of the algorithm;

Sampling a neighbourhood without replacement. This is motivated by the fact that at low levels of the control parameter the current configuration is hopefully close to the globally minimal configuration, but most of the computation involves rejecting transitions. Only a small number of improving transitions are likely to exist and finding them may take some time. Sampling without replacement would reduce computation. The polynomial schedule presented above attempts L transitions where L is the size of the largest neighbourhood. Therefore sampling without replacement would guarantee finding a lower cost configuration, if it existed, so long as the control parameter was low enough to reject all increasing cost transitions. If no transitions are accepted then the current configuration is at least a locally minimal configuration;

Combining simulated annealing with other methods. Simulated annealing can be used to either provide a good starting configuration for another method or improve upon a configuration found by another method. The former situation might be where simulated annealing generates a starting point for a branch and bound exact algorithm. The latter requires that the initial control parameter value be lower than normal otherwise the starting configuration's good features are quickly lost as cost increasing transitions are possibly accepted;

Parallel implementation. The implementation of a parallel version of simulated annealing can be guided by the problem under consideration or by general strategies to take advantage of parallel processing. A possibility is for each processor to perform its own simulated annealing algorithm in isolation and to take the best solution over all processors. An improvement on this is to use the processors' abilities to communicate and share information. For instance all processors can start a new sequence of transitions with the best configuration over all processors found in the previous sequence. The literature on parallel implementations and experiments is growing rapidly. The interested reader is referred to Aarts and Korst [1] and Laarhoven and Aarts [24].

[10] Reeves [31] contains a more recent survey of simulated annealing including enhancements and modifications. Changes to the acceptance probability, cooling schedule, neighbourhood, sampling, cost function, using simulated annealing with other methods and parallel implementations are discussed.

Chapter 6

Simulated Annealing and the Travelling Salesman Problem

This chapter demonstrates and investigates simulated annealing when applied to the familiar travelling salesman problem. This is a precursor to the application of simulated annealing to the Euclidean Steiner tree problem in Chapter 7. The first section of this chapter provides an empirical analysis of the behaviour of the polynomial cooling schedule described in Section 5.4. The run times and quality of solutions using different cooling schedule parameters are investigated. The second section compares simulated annealing, using the polynomial cooling schedule, with a tailored heuristic for the travelling salesman problem.

6.1 An Empirical Analysis of the Polynomial Cooling Schedule

6.1.1 The travelling salesman problem

To be able to use simulated annealing to find good solutions for the travelling salesman problem it is necessary to describe a configuration, a neighbourhood or neighbour generation mechanism and a cost function.

Laarhoven [25] uses a *cyclic permutation* π to describe a configuration or tour. The k^{th} element, $\pi(k)$, of a cyclic permutation gives the successor of city k in the tour represented by permutation π.[1] Although a problem may be symmetric, using a permutation introduces direction. A tour can be described by two permutations (see Figure 6.1). For a symmetric problem with n cities there are $\frac{1}{2}(n-1)!$ possible configurations. The cost function requires a distance or cost matrix giving the distance between every pair of cities. If the distance between cities i and j is $d(i,j)$ then the cost of configuration i is given by $\sum_{k=1}^{n} d\left(k, \pi_i(k)\right)$.

[1] The permutation is of the integers $1, \ldots, n$ where n is the number of cities. $\pi^m(l)$ is shorthand for $\overbrace{\pi(\pi(\ldots\pi(l)))}^{\text{m times}}$. $\pi^m(l)$ is the m^{th} city visited after l in the tour represented by π. The cyclic nature of the permutation is enforced by $\pi^m(l) = l$ for $m = n$ only.

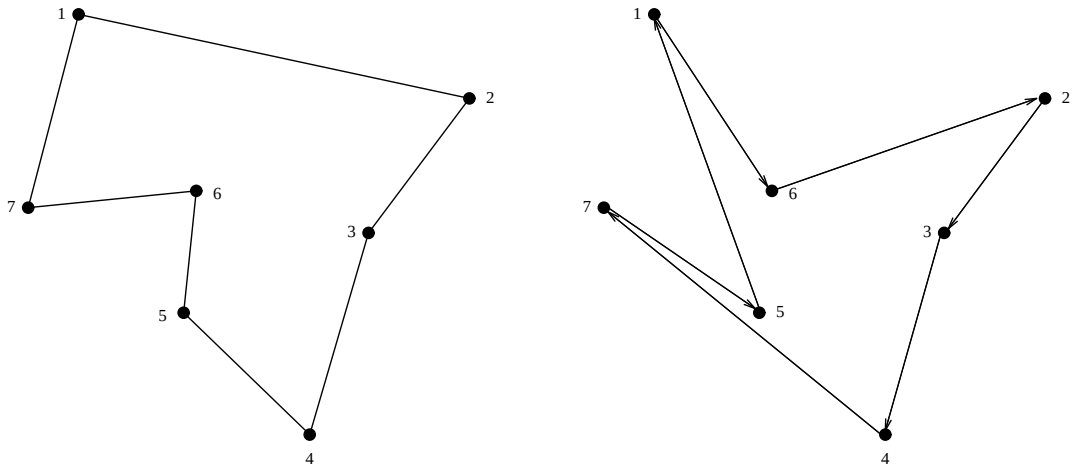


Figure 6.1. An example of cyclic permutations. The tour on the left can be described by cyclic permutations $\{2,3,4,5,6,7,1\}$ or $\{7,1,2,3,4,5,6\}$, the clockwise and counter-clockwise tours respectively. The permutation $\{6,3,4,7,1,2,5\}$ describes the tour shown on the right. If a symmetric problem then the permutation $\{5,6,2,3,7,1,4\}$ describes the equivalent reverse tour.

A simple neighbour generation mechanism for the travelling salesman problem is called the 2-change transition. A 2-change involves replacing two edges in a tour with two edges not in the tour.[2] A 2-change requires a pair of cities l and $\pi^m(l)$ $m \neq n$. The 2-change reverses the order of visiting cities between the pair.[3] An example of a 2-change is shown in Figure 6.2. A 2-change with $m = 1$ results in no change in the tour because there are no cities between l and $\pi(l)$ to be visited in reverse order. A 2-change with $m = 2$ for a symmetric problem also gives no change because only one city is between l and $\pi^2(l)$, namely $\pi(l)$, and reversing the order of visiting $\pi(l)$ does not actually change the tour. The sequence is still l, $\pi(l)$, $\pi^2(l)$.

A neighbour generation mechanism must satisfy two conditions for the simulated annealing algorithm to asymptotically converge to a globally minimal configuration (see Section 5.3.2). The two conditions are:

- Either all generation matrices are symmetric, or are given by the uniform distribution over the neighbourhoods of each configuration;[4]
- A finite number of transitions must exist with which to transform any configuration to any other configuration.

The first condition is satisfied if l takes values in the range $1,\ldots,n$ and m takes values in either the range $1,\ldots,n-1$, or the range $3,\ldots,n-1$. In the

[2] A 2-change is a special case of the k-change used in the k-opt heuristic of Lin and Kernighan [26]. The part of a tour from city l to city $\pi(l)$ is called an edge.

[3] *This definition of a 2-change is applied only to a symmetric and undirected travelling salesman problem in this chapter.* In an asymmetric or directed problem a 2-change does much more than simply replace two edges.

[4] The generation matrices give the probability of generating configuration j given current configuration i (see Section 5.3.1).

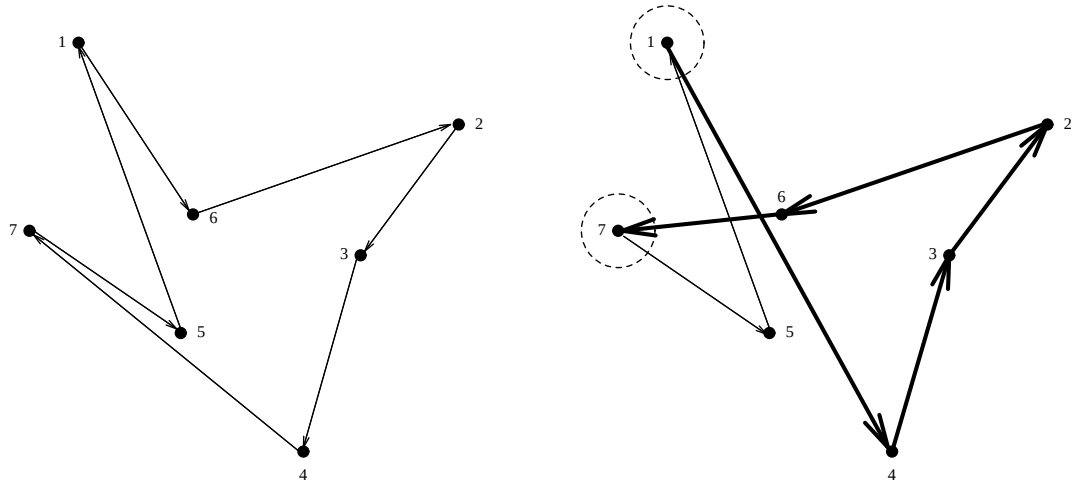


Figure 6.2. An example of a 2-change. The left tour is before the 2-change and the right tour after the 2-change. The pair of cities is 1 and 7 given by $l = 1$ and $m = 5$ (city 7 is the fifth city visited after city 1 in the tour). The order of visiting cities 6, 2, 3 and 4 is reversed.

former case the generation matrices are symmetric but *not* uniform over the neighbourhoods of configurations. In the latter case the matrices are symmetric *and* uniform. Laarhoven [25] uses $1, \ldots, n-1$ for symmetric travelling salesman problems. This gives a neighbourhood size of $\frac{1}{2}n(n-1)$. The $m = 1$ and $m = 2$ transitions result in $2n$ of the $n(n-1)$ possible transitions giving the current configuration. The remaining $n(n-1) - 2n = n(n-3)$ give transitions to the $\frac{1}{2}n(n-3)$ unique configurations. The distribution of neighbours is not uniform over the neighbourhood, but the generation matrices are symmetric. Limiting m to $3, \ldots, n-1$ removes transitions to the current configuration, and reduces the neighbourhood size to $\frac{1}{2}n(n-3)$. The generation matrices are both uniform over the neighbourhoods and symmetric.[5]

Which range for m is better? An important feature, or idea, of simulated annealing is the sampling of the neighbourhoods in search of lower cost configurations. Allowing transitions that generate the current configuration is contrary to this idea. Therefore the $3, \ldots, n-1$ range for m is used for generating 2-changes for the travelling salesman problem.

The second condition for the generating mechanism is that it is possible to move from any configuration to any other using a finite number of transitions. Laarhoven [25] proves this is possible with 2-changes. Appendix E describes the method of creating the finite sequence of transitions and gives an example.

6.1.2 The initial control parameter value

An analysis of the behaviour of the initial control parameter value is presented separately to the full analysis of a later section because the initial control pa-

[5] An example of applying two different 2-changes to the same permutation and giving identical tours is given in Appendix D.

rameter value is the starting point of the simulated annealing algorithm, and therefore of importance. Understanding its behaviour as a function of the initial acceptance ratio and the supposedly arbitrary number of iterations to determine the initial value is essential before moving on to study the behaviour of the polynomial cooling schedule in its entirety (when applied to the travelling salesman problem).

The initial control parameter is determined by an iterative process. A sequence of values are generated by attempting transitions starting from a arbitrary control parameter value. The sequence quickly converges to the initial control parameter value. The equation to update the value is

$$c = \frac{\overline{\triangle C}^{(+)}}{\ln\left(\frac{m_2}{m_2\phi_0 - m_1(1-\phi_0)}\right)}$$

where $\overline{\triangle C}^{(+)}$ is the average increase in cost of *proposed* increasing cost transitions, m_1 is the number of decreasing cost transitions, m_2 is the number of *proposed* increasing cost transitions, and ϕ_0 is the initial acceptance ratio. The word proposed is used because not all increasing cost transitions are accepted by the annealing algorithm, but it is all suggested increasing transitions that are of interest in determining the first control parameter value. The initial acceptance ratio is one of the parameters of the polynomial cooling schedule (see Section 5.4). The process ceases when m_0 transitions have been attempted. Figure 6.3 shows the convergence of the initial control parameter sequences for a one hundred city Euclidean travelling salesman problem.[6] The sequences all quickly stabilise about the true initial value. The travelling salesman problem simulated annealing program is in Appendix F.

The dependence of the initial control parameter on the initial acceptance ratio

The initial control parameter is inversely proportional to the natural logarithm of the reciprocal of an affine function of the initial acceptance ratio. However it is convenient to consider the relationship between the initial control parameter and the initial rejection ratio, the complement of the initial acceptance ratio.

Consider the calculation of the initial control parameter value using an initial acceptance ratio $\phi = 1 - r$ and $\phi' = 1 - kr$ where r is the initial rejection ratio and k is some scaling factor such that $0 < 1 - kr < 1$. After M iterations of the method for determining c_0 the number of proposed cost increasing transitions are m_2 and m_2', and the average proposed increase costs are $\overline{\triangle C}^{(+)}$ and $\overline{\triangle C}^{(+)'}$ respectively. The ratio of c_0' to c_0 is

$$\frac{\overline{\triangle C}^{(+)'} \ln\left(\frac{m_2}{m_2(1-r)-(M-m_2)r}\right)}{\overline{\triangle C}^{(+)} \ln\left(\frac{m_2'}{m_2'(1-kr)-(M-m_2')kr}\right)}.$$

[6] Points are randomly distributed in the unit square.

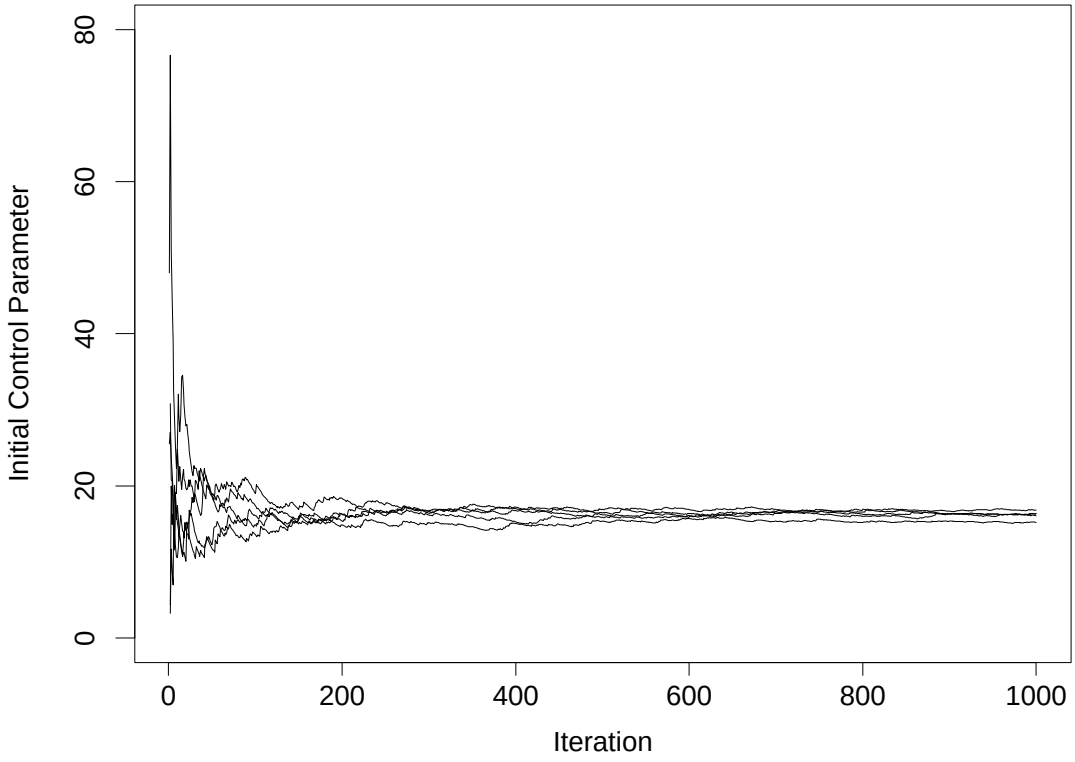


Figure 6.3. An example of the convergence of the initial control parameter value sequence. Five sequences are shown using the same one hundred city Euclidean travelling salesman problem, but different random numbers in the neighbour generation mechanism and annealing acceptance test.

This is transformed into a series in terms of r with the assistance of a symbolic computation package. The series is

$$\frac{\overline{\triangle C}^{(+)'}(M-2m_2)m_2'}{\overline{\triangle C}^{(+)}k(M-2m_2')m_2} + \frac{\left(\frac{M}{2m_2} - \frac{(M-2m_2)^2m_2'}{2m_2^2k(M-2m_2')} - 1\right)}{\overline{\triangle C}^{(+)}}\overline{\triangle C}^{(+)'}r + O(r^2).$$

Clearly the change in the initial control parameter value given a change in the initial rejection ratio is not a simple function of the scaling factor k. However if it is assumed the evolution of the sequences are similar, namely $\overline{\triangle C}^{(+)} \approx \overline{\triangle C}^{(+)'}$ and $m_2 \approx m_2'$, then the approximate ratio of c_0' to c_0 is

$$\frac{1}{k} + \frac{(k-1)(M-m_2)}{2km_2}r + O(r^2).$$

For small initial rejection ratios a change in the initial rejection ratio by a factor of $\frac{1}{k}$ gives an increase in the initial control parameter by a factor of approximately k. This is observed in Figure 6.4. Initial acceptance ratios of 0.9 and 0.99 give initial control parameter values of 1.77 and 16.41 respectively at the 1,000th iteration. The initial rejection ratio changes by a factor of $\frac{1}{10}$, from 0.1 to 0.01, and the initial control parameter value increases by a factor of 9.3. Similarly, for initial acceptance ratios of 0.99 and 0.999, the increase in the initial control parameter is by a factor of 9.6.[7]

Convergence and a simple stopping rule

The initial control parameter value is dependent on problem instance. For example two instances of the travelling salesman problem with the same number of cities will give different values, and a problem with costs or distances scaled up or down will give a different initial value to the original problem. This dependence is primarily due to the presence of $\overline{\triangle C}^{(+)}$ in the above equation. If the initial value is so strongly dependent on the problem details then what is an appropriate value for m_0?

- Too small and the sequence may not have converged, and the initial value could be either lower or higher than the true value. A low value results in the simulated annealing algorithm being prematurely trapped and proceeding to find a locally optimal solution. A high value can result in wasted computation because work is done by the algorithm at control parameter values higher than necessary;
- If m_0 is too large then time is spent needlessly iterating when the sequence has already converged.

The second situation is the lesser of the two evils. But how large is large enough, $m_0 = 1,000$, $m_0 = 1,000,000\ldots$? This question is investigated by

[7] The empirical analysis uses just one sequence of values at each level of the initial acceptance ratio. Further, each sequence is terminated at the 1,000th iteration. Ideally, several longer sequences at each level should be analysed.

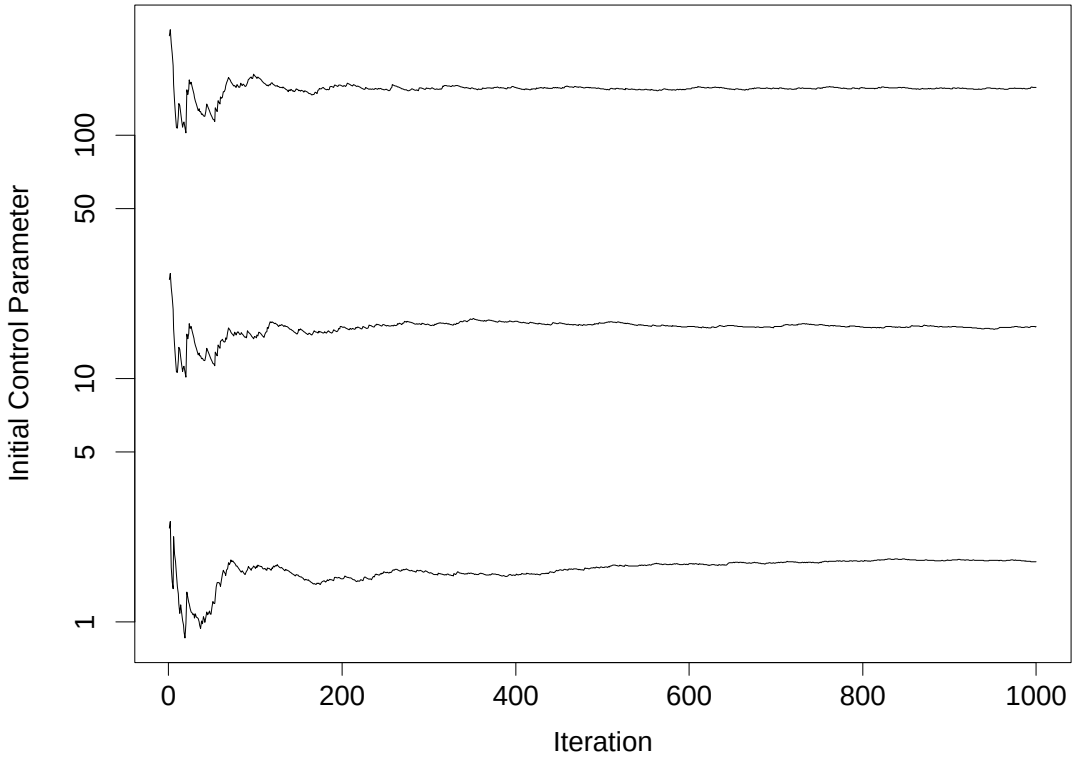


Figure 6.4. The dependence of the initial control parameter on the initial acceptance ratio. The three sequences (from top to bottom) are for initial acceptance ratios of 0.999, 0.99 and 0.9, and each is generated using the same problem and sequence of random numbers.

side-stepping the problem with m_0 and instead considering convergence and a simple stopping rule.

The sequences in Figure 6.3 suggest for the particular problem and initial acceptance ratio parameter used a m_0 of around 600 is acceptable. But this might not be appropriate for another problem. A different approach is to use a simple stopping rule. A possibility is to stop if the current value c_i is within $\pm\lambda\%$ of the last value, that is stop if

$$\left|\frac{c_i - c_{i-1}}{c_{i-1}}\right| < \lambda.$$

The parameter λ is called the *initial stopping tolerance*.[8] The iteration i at which the rule is first satisfied is equivalent to m_0, and c_i is c_0. More elaborate rules can easily be constructed, for example using some smoothed value of recent c_i to compare to the current value or a rule based on statistical confidence intervals, however the above rule is simple and easily implemented and its performance is analysed in the remainder of this section.[9]

To investigate the initial control parameter's dependence on λ and ϕ_0 one hundred runs of the method to determine c_0 were performed for $\phi_0 = 0.9, 0.99, 0.999$ with the same one hundred city Euclidean travelling salesman problem. From each run c_i and i were found at which the stopping rule was first satisfied for $\lambda = 0.01\%, 0.1\%, 1\%$. In addition the true c_0 was found by performing 100,000 iterations for each ϕ_0 ten times.

Table 6.1 shows the results of the runs. The mean and *sample* standard deviation are shown for the true initial value, the estimated initial value and the number of iterations necessary to find the estimate. Figures 6.5 and 6.6 show the distributions of the estimated value and number of iterations for each of the nine combinations of ϕ_0 and λ.

Both the table and figures show that the better the tolerance, that is the smaller, the closer the estimated value is likely to be to the true value. In all cases the mean of the estimated value is less than the true value.[10] For $\phi_0 = 0.9$ the 1% stopping tolerance estimated value is about 30% below the true value and the 0.01% value is about 10% below. Similarly for $\phi_0 = 0.999$ the 1% stopping tolerance estimated value is about 20% below the true value and the 0.01% value is about 5% below. The discrepancy between the estimated and true initial control parameter values decreases with increasing ϕ_0 and decreasing λ. The number of iterations needed to find the estimated value does not change dramatically with ϕ_0, for example for $\lambda = 0.01\%$ the number of iterations is 242 and 261 for ϕ_0 equal to 0.9 and 0.999 respectively. However, what must not be forgotten is that starting the simulated annealing algorithm with a high initial control parameter greatly increases the computation, but with the prospect of obtaining a better solution.[11]

[8] This is different to the stopping parameter ϵ_s used in the polynomial schedule.

[9] The rule for terminating the simulated annealing algorithm which uses ϵ_s is an example of a more elaborate rule.

[10] In Figure 6.3 the c_i sequences approach the true value from below.

[11] These considerations are the subject of Sections 6.1.3 and 6.2.

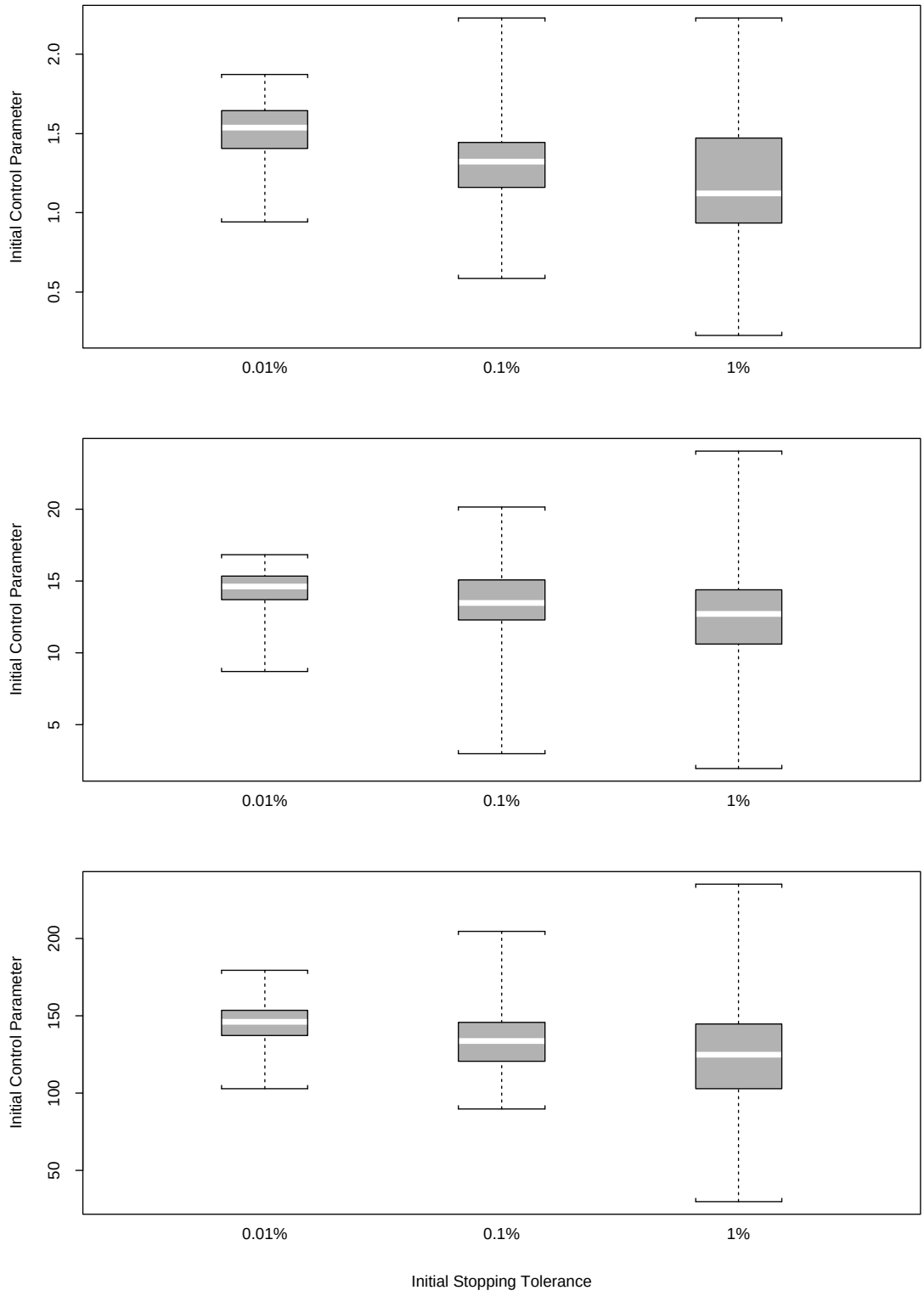


Figure 6.5. The estimated initial control parameter as a function of the initial acceptance ratio and initial stopping tolerance. The top graph is for $\phi_0 = 0.9$, middle for $\phi_0 = 0.99$, and bottom for $\phi_0 = 0.999$. Each boxplot shows the distribution of c_i. The white area within the shaded box is the median value, the shaded box shows the inter-quartile range, and the "whiskers" top and bottom show the extreme values.

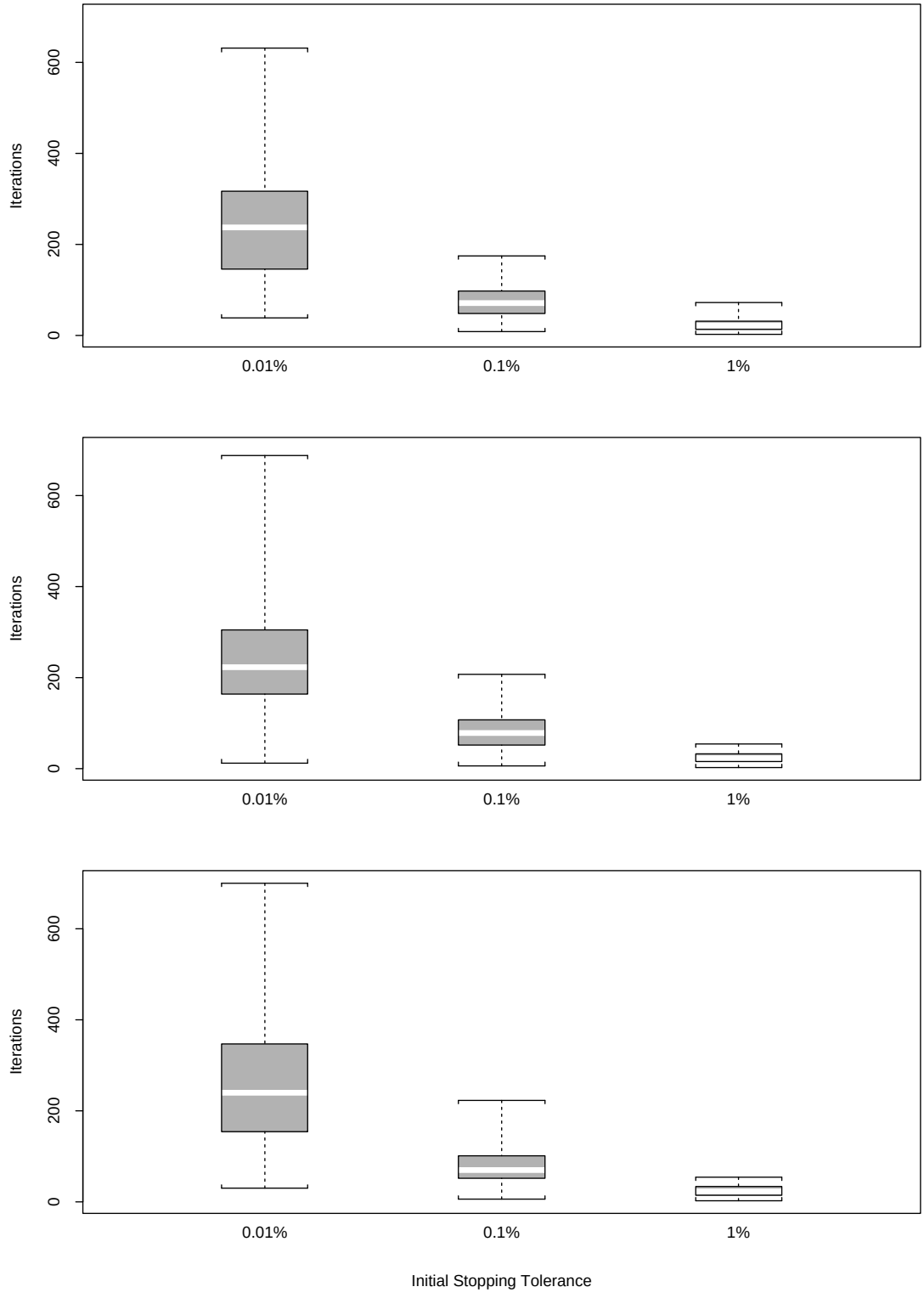


Figure 6.6. The iterations to find the estimated initial control parameter as a function of the initial acceptance ratio and initial stopping tolerance. The top graph is for $\phi_0 = 0.9$, middle for $\phi_0 = 0.99$, and bottom for $\phi_0 = 0.999$.

Initial Acceptance Ratio ϕ_0	True Initial Control Parameter c_0		Initial Stopping Tolerance λ	Estimated Initial Control Parameter c_i		Number of Iterations i	
	Mean	Standard Deviation		Mean	Standard Deviation	Mean	Standard Deviation
0.900	1.68	0.00	0.01%	1.51	0.19	242.13	131.15
			0.10%	1.32	0.27	75.08	38.85
			1.00%	1.17	0.43	23.56	12.35
0.990	15.37	0.00	0.10%	14.28	1.64	237.85	126.81
			0.10%	13.31	2.69	82.09	41.67
			1.00%	12.51	3.66	25.50	12.67
0.999	152.19	0.25	0.01%	144.97	12.83	261.45	135.30
			0.10%	133.72	23.23	78.66	43.28
			1.00%	123.13	33.61	25.16	12.85

Table 6.1. Initial control parameter empirical data. For each initial acceptance ratio one hundred runs of the method of determining the initial control parameter were performed. For each run the estimated control parameter values were found using the stopping rule with different initial stopping tolerances. The true value was found by running the method ten times for a large number of iterations. The same one hundred city Euclidean travelling salesman problem was used in each run.

The effort needed to find the initial value is determined by λ, and this effort is negligible compared to that in the simulated annealing algorithm proper. The number of iterations is likely to be a fraction of the number of transitions attempted in a chain. For instance, no more than about 600 iterations are required to find c_0 in the empirical analysis above, and the number of transitions in a chain for this problem is $\frac{1}{2}100(100-3) = 4,850$. The primary decision problem is deciding upon ϕ_0, and once determined the initial control parameter should be found using as small a λ as possible to ensure repeated runs of the simulated annealing algorithm start from more or less the same initial control parameter value.

An alternative to using a simple stopping rule

The above method of finding c_0 introduces another parameter, the initial stopping tolerance, and the results show that even for small tolerances the estimated initial control parameter value is less than the true value. Neither is desirable. Further, the computational effort to find the initial value is relatively insignificant. So why not try a little harder to get a better initial value?

An alternative is to use a fixed number of iterations for a problem, but a number that varies with problem size and takes account of the nature of the problem. The number of transitions parameter L of the polynomial schedule is such a quantity. It is the size of the largest neighbourhood. For the same one hundred city problem ten runs at each value of ϕ_0 were performed for 4,850 iterations. The results are shown in Table 6.2. The results show the means of the estimated values are all within about 1% of the true values. Further the sample standard deviations are small (from a sample of size 10). The same analysis is performed for a two hundred city problem where L is $\frac{1}{2}200(200-3) = 19,700$. The true initial values are determined using 400,000 iterations. The

results are shown in Table 6.3.

Initial Acceptance Ratio ϕ_0	True Initial Control Parameter c_0		Estimated Initial Control Parameter $c_{4,850}$	
	Mean	Standard Deviation	Mean	Standard Deviation
0.900	1.68	0.00	1.66	0.02
0.990	15.37	0.00	15.45	0.26
0.999	152.19	0.24	152.05	2.62

Table 6.2. Estimated initial control parameter empirical data using the number of transitions for a one hundred city problem.

Initial Acceptance Ratio ϕ_0	True Initial Control Parameter c_0		Estimated Initial Control Parameter $c_{19,700}$	
	Mean	Standard Deviation	Mean	Standard Deviation
0.900	1.72	0.00	1.72	0.01
0.990	15.95	0.02	15.94	0.11
0.999	157.97	0.17	158.26	0.89

Table 6.3. Estimated initial control parameter empirical data using the number of transitions for a two hundred city problem.

Using L is a convenient and accurate method of finding c_0 at the expense of a longer execution time compared to the stopping rule with λ, and is used in the remainder of this chapter and Chapter 7.

6.1.3 The empirical analysis proper

A variety of experiments are performed using the polynomial cooling schedule and randomly generated symmetric travelling salesman problems. The random problems are used to investigate the behaviour of the polynomial schedule and its influence on the quality of solution and the execution time.

A preliminary investigation

Figure 6.7 shows the evolution of the control parameter and the smoothed average cost for each chain for each of five runs of the simulated annealing algorithm on the same randomly generated twenty city Euclidean travelling salesman problem.[12] Typical behaviour of c_k and $\overline{\mu}_k$ is observed. The control

[12]The cities have x and y coordinates randomly distributed in the interval (0,100). The smoothed value is found using a simple five point moving average. The polynomial schedule parameters are: $\phi_0 = 0.9$, $\delta = 0.1$ and $\epsilon_s = 10^{-6}$.

parameter slowly reduces until a point is reached at which few changes occur, and the standard deviation of the configuration costs in a chain sharply decreases. This results in an equally sharp drop in the control parameter.

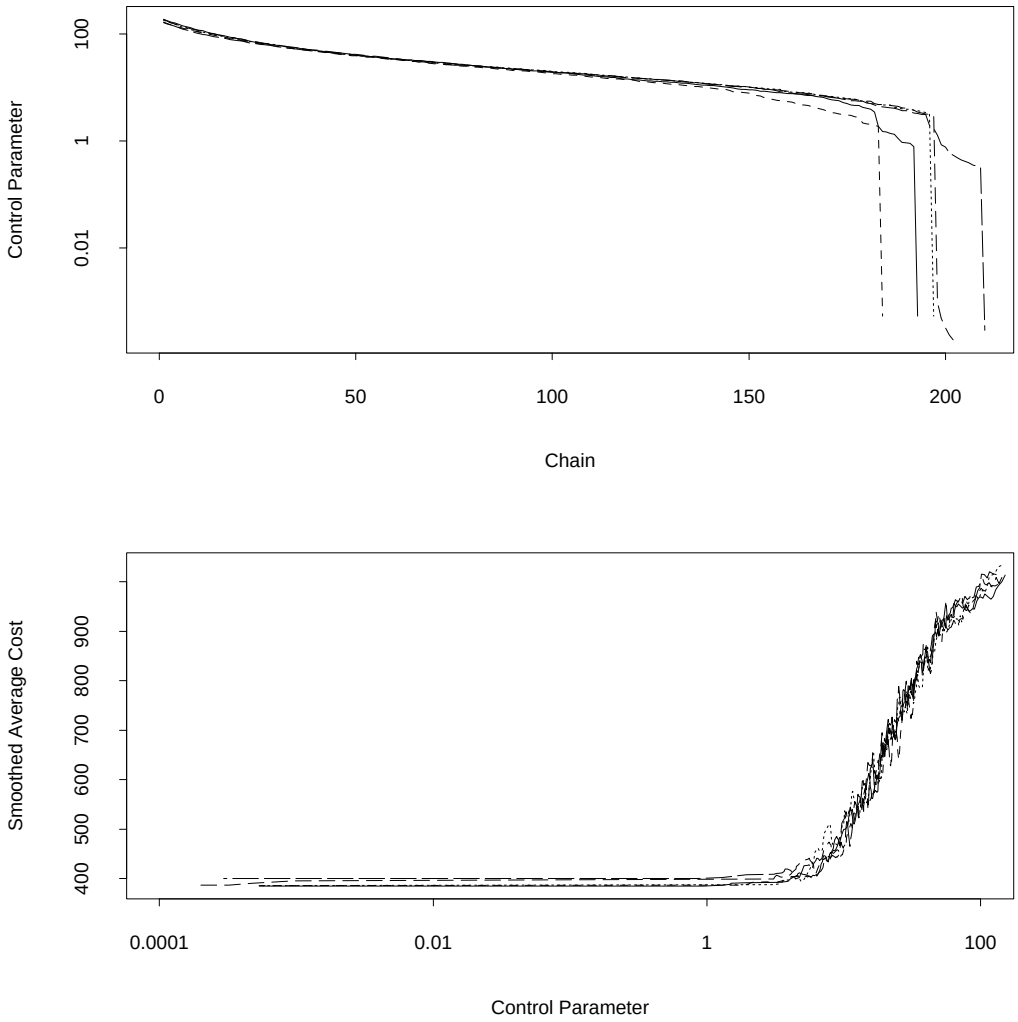


Figure 6.7. The top graph shows the evolution of the control parameter for five runs of the simulated annealing algorithm for a twenty city Euclidean travelling salesman problem. The bottom graph shows the smoothed average cost $\overline{\mu}_k$ as a function of the control parameter for the same runs.

An assumption of the derivation of the decrement rule for the control parameter is that the distribution of costs in a chain can be approximated by the normal distribution (see Section 5.4.1). Figure 6.8 shows boxplots of every fifth chain's configuration costs from one run of the simulated annealing algorithm for a random fifty city problem. Figure 6.9 shows the same information in the form of normal probability plots. The assumption of normality is given some credence by the plots. The distributions of early chains, that is at high control parameter values, are approximately normal, with perhaps slightly longer tails

than the normal. At low levels of the control parameter the distributions do not appear to be normal.

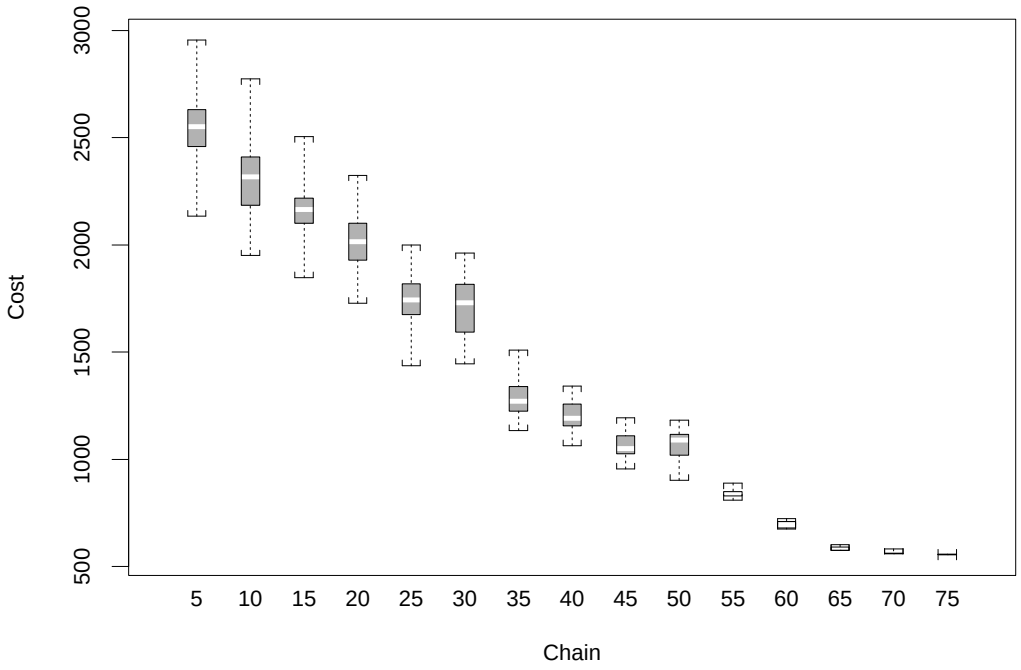


Figure 6.8. The boxplots of the configuration costs for every fifth chain from one run of the simulated annealing algorithm for a fifty city Euclidean travelling salesman problem.

A further assumption is that $\frac{\mu_k - C_{i*}}{\mu_1 - C_{i*}}$ and $\frac{\sigma_k}{\sigma_1}$ as functions of c_k are similar for small δ, where C_{i*} is the cost of a globally optimal configuration. This is confirmed in Figure 6.10. The two ratios are plotted side-by-side for each of three runs for a fifty city problem with $\delta = 0.1$. The general shape of the two functions are similar.

The investigation of the behaviour of the polynomial cooling schedule

To investigate the behaviour of the polynomial cooling schedule ten runs of the simulated annealing algorithm were performed on three instances of the Euclidean travelling salesman problem with fifty, one hundred and two hundred cities. For each set of ten runs different schedule parameters were used. A base set of parameters was used and parameters were varied one at a time from this base set. The base set of parameters is $\phi_0 = 0.99$, $\delta = 1$ and $\epsilon_s = 10^{-5}$, and parameters can take on the following values: $\phi_0 \in \{0.9, 0.99, 0.999\}$, $\delta \in \{0.1, 1, 10\}$ and $\epsilon_s \in \{10^{-7}, 10^{-5}, 10^{-3}\}$. For example, to investigate the behaviour of the schedule with respect to ϕ_0 three sets of ten runs were performed with $\{\phi_0 = 0.9, \delta = 1, \epsilon_s = 10^{-5}\}$, $\{\phi_0 = 0.99, \delta = 1, \epsilon_s = 10^{-5}\}$ and $\{\phi_0 = 0.999, \delta = 1, \epsilon_s = 10^{-5}\}$.

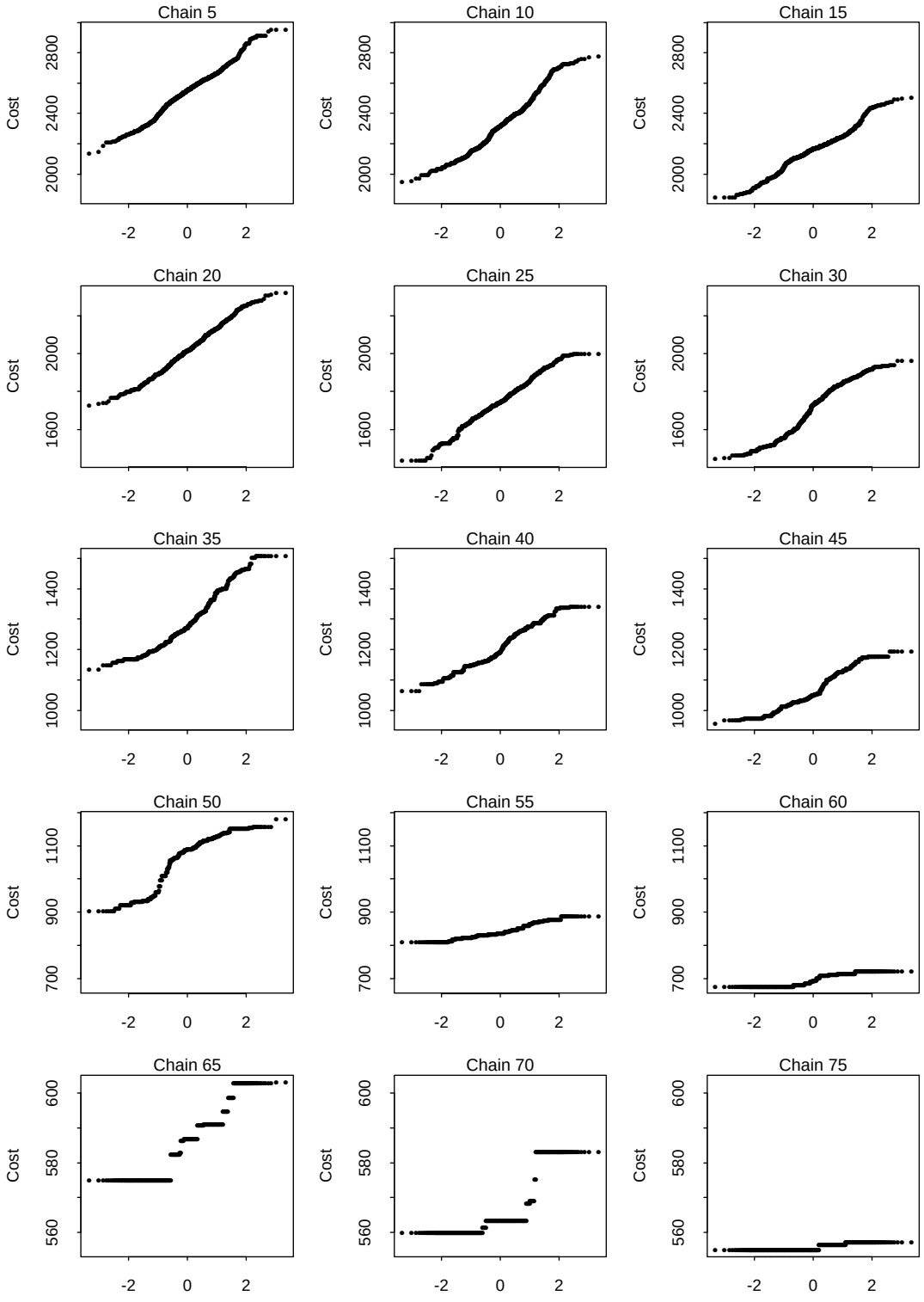


Figure 6.9. The normal probability plot of the configuration costs for every fifth chain from one run of the simulated annealing algorithm for a fifty city Euclidean travelling salesman problem. The x-axis of each plot is the standard normal variate. A straight line indicates the distribution of the configuration costs is normal. An "S" shape indicates the distribution has longer tails than the normal distribution.

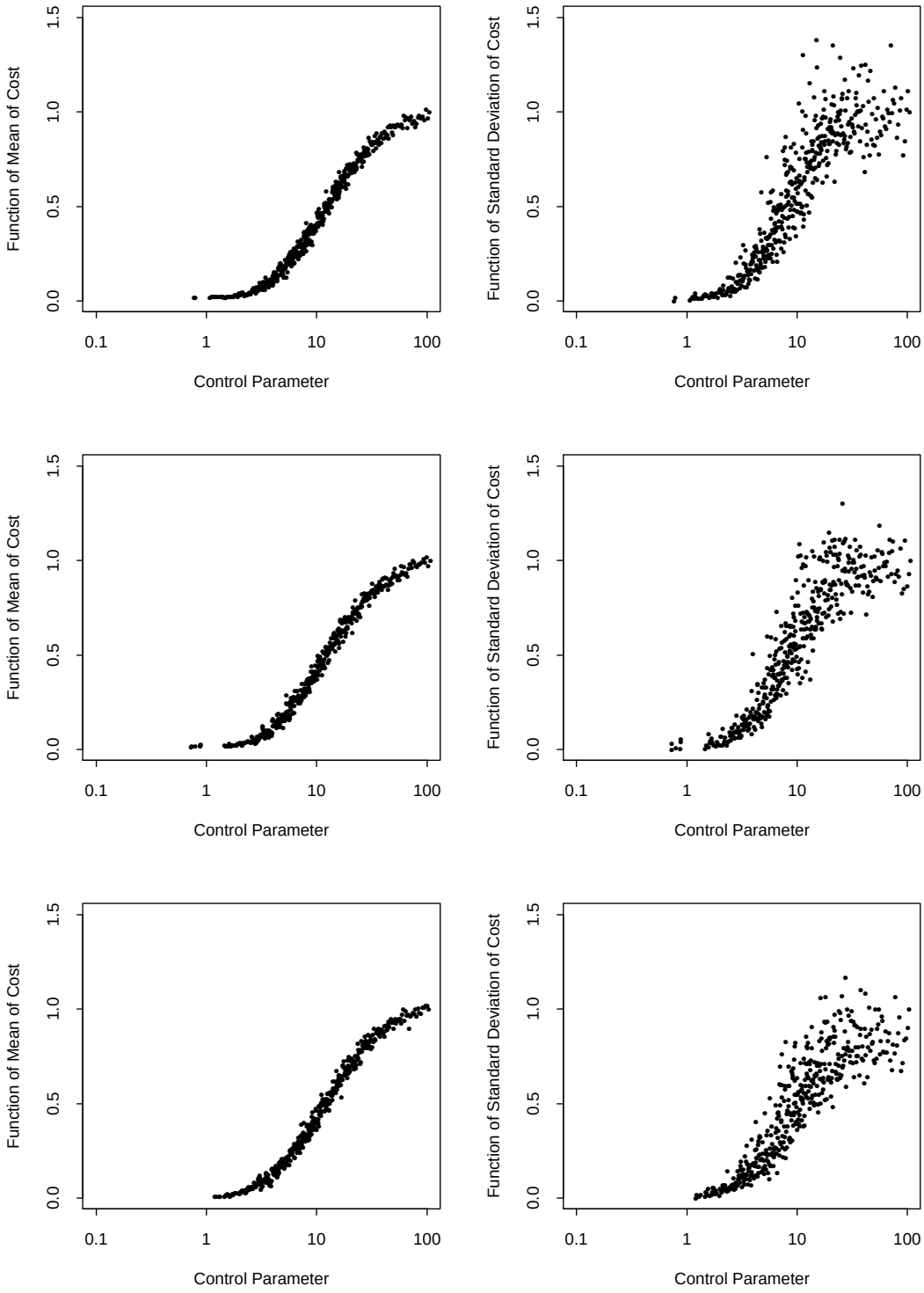


Figure 6.10. The left hand graphs are plots of the function $\frac{\mu_k - C_{i^*}}{\mu_1 - C_{i^*}}$ where C_{i^*} is the cost of a globally optimal configuration. The right hand graphs are plots of the function $\frac{\sigma_k}{\sigma_1}$. Each row of graphs is from the same run of the simulated annealing algorithm. All three runs are for the same Euclidean travelling salesman problem instance.

The execution times and final solution costs were analysed. The costs were compared to the optimal cost by computing the percentage the final solution's cost is above the optimal cost. The optimal cost was approximated by the best solution found from many runs of the k-opt heuristic with randomly generated starting points (see Section 6.2). This approximate optimal solution is called the near-optimal solution.

Figure 6.11 shows the distribution of each set of runs percentage above near-optimal when varying each parameter for the fifty, one hundred and two hundred city problems. The most important feature of the plots is that the distance parameter δ is the primary factor in determining the quality of solution obtained from the simulated annealing algorithm. A small δ gives a solution more likely to be close to the near-optimal solution. Unfortunately there is no evidence that the closeness decreases with increasing problem size. The other two parameters do not influence the quality of solution to the same extent, if at all.

A similar analysis of the execution times reveals that δ is again the determining factor, and within a set of ten runs there was very little variation. However execution times do grow rapidly with increasing problem size and decreasing δ. Table 6.4 shows the mean and sample standard deviations of the percentages above near-optimal and execution times for the three problems and levels of δ.

Problem Size	Distance Parameter δ	Percentage above Near-Optimal		Execution Time (seconds)	
		Mean	Standard Deviation	Mean	Standard Deviation
50	0.1	0.69	0.67	53.98	1.11
50	1.0	1.04	0.83	8.19	0.25
50	10.0	4.12	3.46	3.09	0.21
100	0.1	1.04	0.63	662.37	9.06
100	1.0	3.32	1.49	96.55	2.36
100	10.0	4.53	2.23	32.83	1.25
200	0.1	1.31	1.07	8597.45	310.30
200	1.0	3.79	1.57	1241.14	60.48
200	10.0	4.40	1.81	398.64	7.45

Table 6.4. Simulated annealing empirical data when using the polynomial cooling schedule on randomly generated Euclidean travelling salesman problems.

The analysis of the quality of solution and execution times indicates that care should be exercised in choosing δ, and a trade-off between quality of solution and execution time exists. Using a high initial acceptance ratio and small stopping parameter does not significantly increase execution time and is a safeguard against early convergence to a local optimum and premature termination of the algorithm.

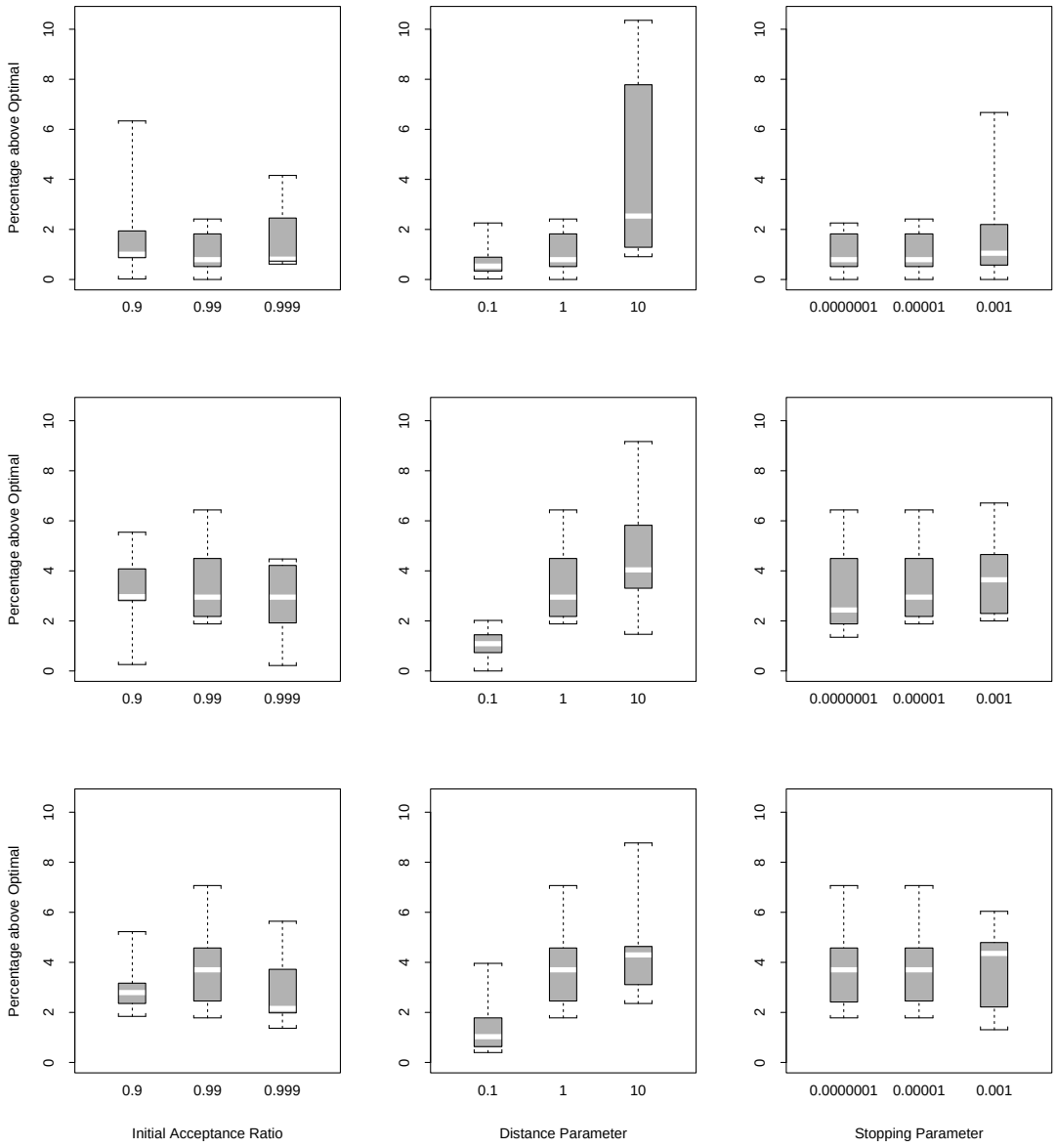


Figure 6.11. Each row (from top to bottom) shows boxplots of the final solution's cost as a percentage above near-optimal for fifty, one hundred and two hundred city problems respectively. Each column contains boxplots of the percentage when only the one parameter is varied from the base set.

6.2 A Comparison of Simulated Annealing with a Tailored Heuristic for the Travelling Salesman Problem

In this section simulated annealing is compared with the k-opt heuristic of Lin and Kernighan [26] for the travelling salesman problem and a heuristic using repeated application of 2-changes. The execution times and quality of solutions are compared for simulated annealing runs using the polynomial cooling schedule and repeated runs of the k-opt heuristic and 2-changes for different Euclidean travelling salesman problems from the literature with known optimal solutions.

6.2.1 The k-opt and repeated 2-change heuristics

The k-opt heuristic is a very fast iterative improvement method for obtaining good travelling salesman problem solutions. The solutions are often optimal or very close to optimal. Unlike the 2-change heuristic which always replaces two edges in a tour with two edges not in the tour, the k-opt heuristic replaces k edges, a k-change. The number of edges replaced is determined from iteration to iteration and varies. The method for determining k is itself iterative. Possible replacements are considered until no further gain is possible or the benefit in increasing k outweighs the effort in determining the edges involved. The heuristic terminates at a solution at which no further cost decreasing k-changes can be made.

The repeated 2-change heuristic is a very simple method for finding a solution. Starting with an arbitrary solution, 2-changes are considered until a cost decreasing change is found and the solution is changed. The process is repeated until all possible 2-changes have been attempted on the current solution without any change giving a decreasing cost transition. The current solution is the final solution. An alternative implementation is to consider all 2-changes for a solution and accept the change giving the greatest decrease in cost.

6.2.2 The empirical comparison

Six problems from the literature with known optimal solutions are used to compare heuristics. The problems vary in size from 51 cities to 105 cities. The problems are all symmetric Euclidean problems. These test problems are in Appendix G.[13]

[13]The code for the k-opt heuristic was kindly supplied by D Slowinski (`slow@runkel.cray.com`). It is in FORTRAN and was compiled using the Sun FORTRAN compiler. It has been assumed the implementation of the k-opt heuristic is correct. The production of optimal solution costs for the problems with known optimal solutions supports this assumption.

Simulated annealing

Table 6.5 shows the average final solution costs and average percentage above optimal over ten runs of the simulated annealing algorithm with $\delta = 0.1, 1, 10$ for each problem. This shows the same behaviour of the polynomial schedule as in the previous section: a smaller δ gives a better solution. Table 6.6 shows that better solutions are obtained at the expense of greater execution times.

Problem	Distance Parameter δ	Cost		Average Percentage above Known Optimal
		Mean	Standard Deviation	
EIL051	0.1	443.15	5.95	3.33%
EIL051	1.0	444.37	4.90	3.61%
EIL051	10.0	453.06	7.15	5.64%
EIL076	0.1	556.46	6.43	2.22%
EIL076	1.0	565.74	8.98	3.93%
EIL076	10.0	572.91	6.45	5.24%
KRO124	0.1	21445.30	92.06	0.75%
KRO124	1.0	21849.89	295.62	2.65%
KRO124	10.0	21912.51	202.78	2.95%
KRO126	0.1	21041.33	172.44	1.40%
KRO126	1.0	21215.25	360.78	2.24%
KRO126	10.0	21471.22	444.33	3.47%
KRO127	0.1	21492.81	126.22	0.93%
KRO127	1.0	21643.44	165.33	1.64%
KRO127	10.0	22190.73	315.83	4.21%
LIN105	0.1	14566.83	67.09	1.28%
LIN105	1.0	14730.05	276.85	2.41%
LIN105	10.0	15059.14	283.37	4.70%

Table 6.5. Simulated annealing final solution costs and percentages above known optimal. The digits in the problem name are the problem size except for the KRO problems which are in fact one hundred city problems.

The average execution times for the $\delta = 0.1$ runs are the times used for repeated runs of the k-opt and 2-change heuristics. For example, each run of the k-opt and 2-change are allowed 663 seconds of computing time on the KRO127 problem and 57 seconds on the EIL051 problem. For each problem there are ten runs.[14]

[14]The method of implementing the repeated 2-change is to let $l = 1, \ldots, n$ and for every l let $m = 3, \ldots, n-1$. Of the $n(n-3)$ 2-changes only $\frac{1}{2}n(n-3)$ unique solutions are generated. Instead of attempting to detect 2-changes that may already have been attempted the 2-change heuristic was allowed to run for twice the time to compensate for the lack of detection of duplicated proposed 2-changes.

Problem	Distance Parameter δ	Execution Time (seconds)	
		Mean	Standard Deviation
EIL051	0.1	56.96	0.82
EIL051	1.0	8.34	0.39
EIL051	10.0	3.10	0.22
EIL076	0.1	239.06	5.07
EIL076	1.0	40.48	10.21
EIL076	10.0	12.91	0.44
KRO124	0.1	679.41	16.92
KRO124	1.0	114.49	40.67
KRO124	10.0	34.10	1.14
KRO126	0.1	658.05	8.81
KRO126	1.0	97.68	2.32
KRO126	10.0	33.02	1.05
KRO127	0.1	663.05	14.98
KRO127	1.0	95.45	1.95
KRO127	10.0	31.74	1.14
LIN105	0.1	807.35	23.57
LIN105	1.0	119.31	2.89
LIN105	10.0	39.58	1.63

Table 6.6. Simulated annealing execution times.

Problem	Mean Simulated Annealing Cost	Mean *Best* k-opt Cost	Mean Number of Solutions Found	Mean Proportion giving Optimal Cost	Average Percentage above k-opt
EIL051	443.15	428.87	1605	1.3%	3.3%
EIL076	556.46	544.37	3306	8.4%	2.2%
KRO124	21445.30	21285.44	5571	0.3%	0.8%
KRO126	21041.33	20750.76	5330	0.5%	1.4%
KRO127	21492.81	21294.29	4538	0.1%	0.9%
LIN105	14566.83	14383.00	5144	5.0%	1.3%

Table 6.7. k-opt heuristic comparison with simulated annealing. The simulated annealing mean cost is the $\delta = 0.1$ mean. The k-opt means are over ten runs each of length equal to the average simulated annealing execution time for $\delta = 0.1$.

Comparison with k-opt heuristic

Table 6.7 shows the results of the k-opt heuristic runs. For all six problems the k-opt heuristic found the optimal solution some of the time. Therefore the "Average Percentage above k-opt" column in the table shows the same figures as the "Average Percentage above Known Optimal" column in Table 6.5. The k-opt heuristic clearly out-performs simulated annealing.

Comparison with repeated 2-change heuristic

Table 6.8 shows the results of the repeated 2-change runs. For four of the six problems the repeated 2-change heuristic is better than simulated annealing. Although for three of the four the difference is less than 1%. For none of the problems did the 2-change heuristic find the optimal solution. The average number of solutions found in the runs of the 2-change heuristic is significantly lower than with the k-opt heuristic. This is partly due to the superiority of the k-opt method and to its implementation in FORTRAN compared to the 2-change's implementation in C++.

Summary

The results of the above analysis support the conclusions of Laarhoven [25]. He showed that k-opt is far better than simulated annealing. That is, a *good* tailored heuristic for a specific problem will generally out-perform simulated annealing. Laarhoven concludes that simulated annealing is better than repeated application of the generation mechanism used in the simulated annealing algorithm when both are given the same amount of computing time. For example in the case of the travelling salesman problem, simulated annealing using a 2-change is better than repeated 2-changes as used above.

However, the comparison above of simulated annealing and repeated 2-changes favours the latter by a small amount. Laarhoven's analysis is over ten

Problem	Mean Simulated Annealing Cost	Mean *Best* 2-change Cost	Mean Number of Solutions Found	Average Percentage above 2-change
EIL051	443.15	432.27	896	2.5%
EIL076	556.46	560.97	1202	-0.8%
KRO124	21445.30	21418.66	1377	0.1%
KRO126	21041.33	20977.47	1355	0.3%
KRO127	21492.81	21714.56	1374	-1.0%
LIN105	14566.83	14468.78	1394	0.7%

Table 6.8. Repeated 2-change comparison with simulated annealing.

problems (compared to six) and for five of the ten problems only five runs equal in length to the average simulated annealing time are performed (compared to ten).[15] Three of the problems are 120, 318 and 442 city problems and for these simulated annealing is better than repeated 2-changes by 1.25%, 3.13% and 4.50% respectively. The better performance for larger problems is perhaps reason enough to suggest simulated annealing is the better heuristic.

[15]For the remaining five problems ten runs are performed in Laarhoven's investigation.

Chapter 7

Simulated Annealing and the Euclidean Steiner Tree Problem

Earlier chapters have shown that the Euclidean Steiner tree problem is a very difficult problem and that heuristics are the only practical means available for finding good solutions in acceptable amounts of time. In Chapter 6 it was shown that simulated annealing can be successfully applied to the travelling salesman problem. In this chapter an attempt is made to apply simulated annealing to the Steiner tree problem. The necessary components of an annealing approach are discussed, experiments are described and the results analysed.

There appears to be little work on applying simulated annealing to finding Steiner minimal trees. Two papers of most direct relevance are Lundy [27] and Hesser *et al.* [16] (both are discussed briefly in Chapter 4).[1] Lundy uses simulated annealing to find evolutionary trees for n populations by constructing a binary tree linking the populations via $n-2$ ancestral populations. A two stage random transition mechanism is used. Firstly, a full Steiner topology is transformed into a different full topology. Secondly, the $n-2$ Steiner points are sequentially moved to optimal locations (with respect to their immediate neighbours in the topology). Unfortunately, Lundy's discussion of using simulated annealing is limited to finding best connections with a n point full Steiner topology. Using simulated annealing to find the positions of Steiner points given a full topology is now redundant because of the existence of Hwang's linear time algorithm for finding a full Steiner tree for a given topology (see Section 2.9.4).

Hesser *et al.* [16] devote most of their effort to using a genetic algorithm to find Steiner minimal trees. A simulated annealing algorithm is discussed *very* briefly. They give insufficient details on the actual implementation and problems encountered. It is not clear in either the genetic algorithm or annealing discussion how the number of Steiner points in a solution can be changed.

[1] Dowsland [9], Schiemangk [34] and Kapsalis *et al.* [22] are presentations of the application of simulated annealing and genetic algorithms to the *network* Steiner problem.

7.1 Second Thoughts on Using Simulated Annealing

The references cited above provide little guidance on using simulated annealing with the Euclidean Steiner tree problem. A reaction to the small number of references of limited assistance is to ask if using simulated annealing is a fruitless task. An endeavour with no reward and little joy.

The tailored heuristics of Smith *et al.* [37] and Beasley and Goffinet [2] (see Chapter 4) give very good solutions and do so in reasonable amounts of time. Analysis of Cockayne and Hewgill's one hundred point test problems (see Appendix B) shows that most Steiner minimal trees are close to the minimum spanning tree in the sense that the SMT tends to follow the MST. Further, the SMTs are predominately composed of two, three and four point FSTs. These structures are the explicit building blocks of the Smith *et al.* heuristic.[2] Finally, the average reduction given by the SMT over the MST is around 3%, the heuristics also give reductions of around 3%. This indicates that the tailored heuristics do a good job of finding approximate solutions that are close to optimal. And the approach of improving upon the MST by using three and four point FSTs is a correct and successful approach (at least for randomly generated problems).

So is simulated annealing worth pursuing? Perhaps not from a practical problem solving point of view. But there is still the intellectual exercise of implementing a simulated annealing approach to the Euclidean Steiner tree problem. It is certain that simulated annealing will be slower than the tailored heuristics but it is possible that annealing can give better quality solutions if given enough time.

A "pure" form of annealing is described in the following sections. Pure in the sense that little problem specific information is used and very simple transitions from one solution to another are performed. Information provided by the minimum spanning tree and Steiner polygon, important in optimal algorithms, is ignored. The annealing schedule, acceptance rule and cost function work together to give good solutions.

7.2 Solutions and Transition Mechanisms

Fundamental components of a simulated annealing algorithm are a solution, a means of creating a neighbour solution and a cost function. A restatement of the Euclidean Steiner tree problem provides insight into what a solution and cost function are for an annealing implementation. The problem is: given a set of points $\mathcal{A} = \{a_1, \ldots, a_n\}$ in the plane, what is the set of points $\mathcal{S} = \{s_1, \ldots, s_k\}$, $k \geq 0$, that gives the minimal length spanning tree of $\mathcal{A} \cup \mathcal{S}$?

A solution is a set of points, possibly an empty set. It is also known that the maximum size of the set is $n-2$ (see Section 2.5). For the purposes of annealing the cost of a solution is the length of the minimum spanning tree of the given

[2]Beasley and Goffinet do not explicitly use three and four point FSTs but instead use the Delaunay triangulation to generate possible Steiner points.

points and the solution points as a percentage of the minimum spanning tree of the given points only. A neighbour generation mechanism is some random change to a solution. In this case, an addition, deletion or replacement of a point in the solution.

The following three sections describe each type of change. Each is very simple. More sophisticated changes could be used, but simplicity and ease of implementation are more important at this early stage of applying annealing to the Steiner tree problem.

7.2.1 Adding a point

If there are less than $n-2$ points in the current solution $\mathcal{S}$ then another point can be added. The size of the current solution is denoted by k. The added point is the Steiner point of the triangle formed by three distinct randomly selected points from the $n+k$ points in $\mathcal{A} \cup \mathcal{S}$. Further, the added point must not already be in $\mathcal{S}$. It is possible that for many combinations of three points the Steiner point will not exist. Figures 7.1 and 7.2 show an example of adding a point to an existing solution containing five points.

It appears from the above example that added points will often be in ridiculous locations from a Steiner minimal tree point of view. However as the annealing progresses such additions are increasingly likely to *not* be accepted. Instead, additions that give decreasing length minimum spanning trees will be the only changes accepted and the solution will converge to a Steiner tree, preferably the SMT but not necessarily so. An example of the evolution of a solution using annealing is given below.

7.2.2 Deleting a point

It is possible to delete a point in the solution if $k > 0$. A point is selected randomly from $\mathcal{S}$. Figure 7.3 shows the result of deleting point A from the solution in Figure 7.2.

Deletions of "good" Steiner points will occur with decreasing frequency as the simulated annealing algorithm runs. And "bad" Steiner points, those not of degree three or with all angles not equal to 120°, are more likely to be removed.

7.2.3 Replacing a point

Replacing a point is possible if $k > 0$. Replacement is a deletion followed by an addition. It is possible for the solution to be unchanged at the end of the process but this is unlikely. Figures 7.4 and 7.5 show an example of replacing a point.

The replace change combines deletion and addition into one. It provides a means of moving a Steiner point. This is especially important near the end of the annealing when the control parameter is such that nearly all increasing cost moves are rejected. Consider a point that gives a minimum spanning tree with

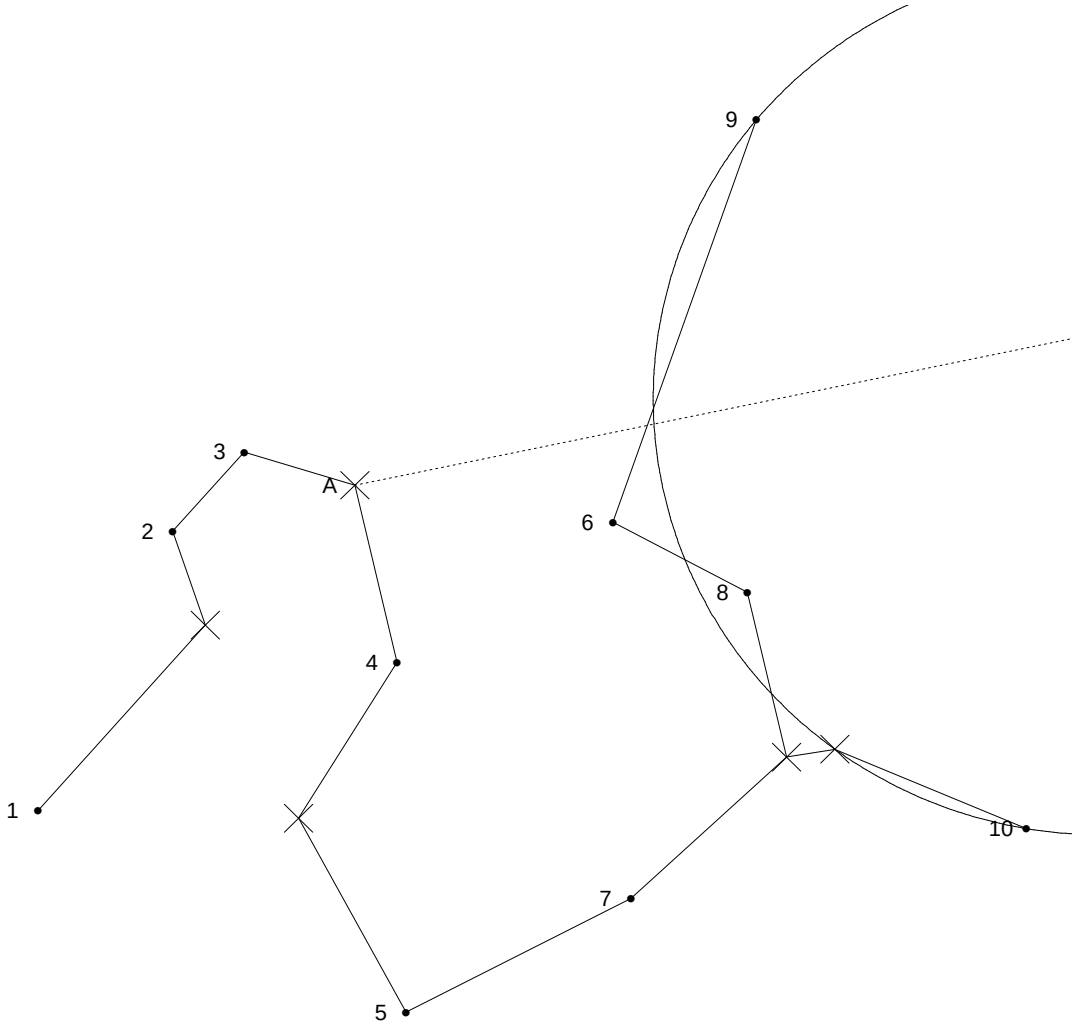


Figure 7.1. A point defined by the three points 9, 10 and A is added to the existing five point solution (marked by crosses). The new point is at the intersection of the circle (partly shown) and the axis of the three point FST (dashed line). The current solution is clearly not optimal because most of the "Steiner points" are of degree two.

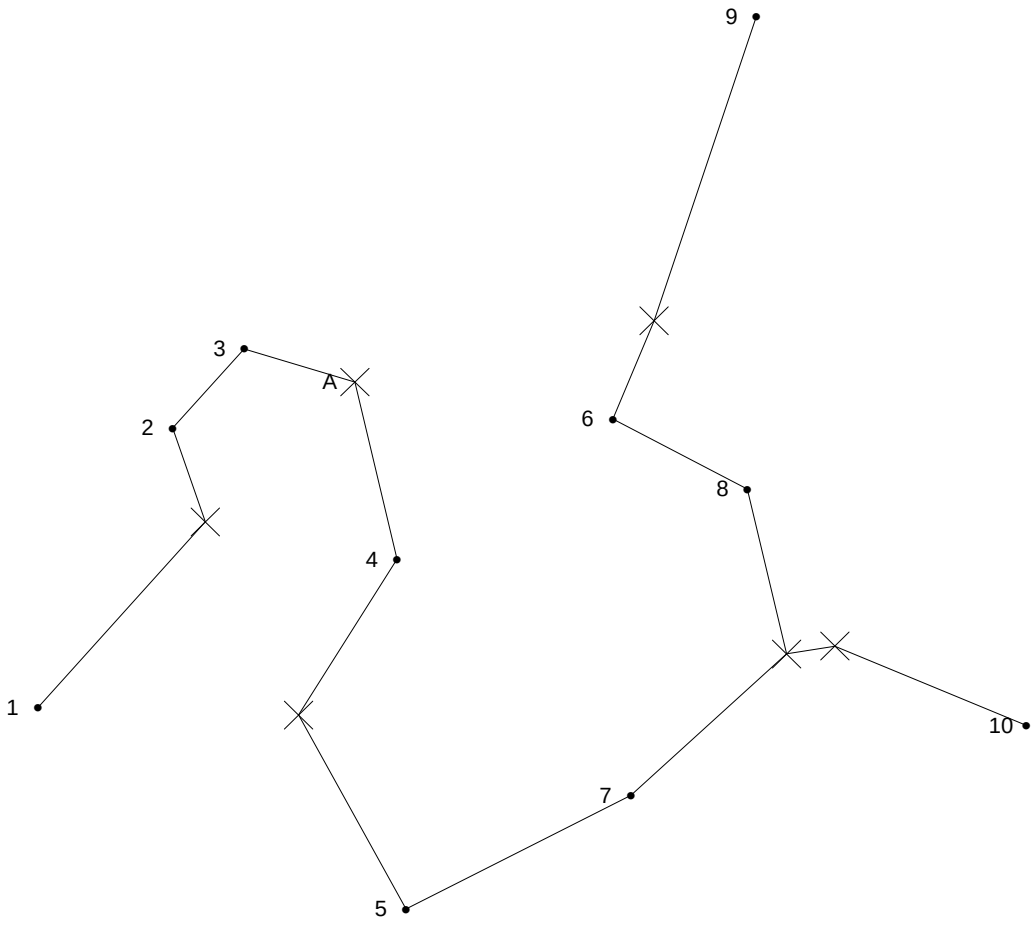


Figure 7.2. The new solution after adding a point. It does little to improve the quality of the solution. The new point is above point 6 and in the new minimum spanning tree is directly connected to 6 and 9. It is not directly connected to two of the points on which it is based, 10 and A.

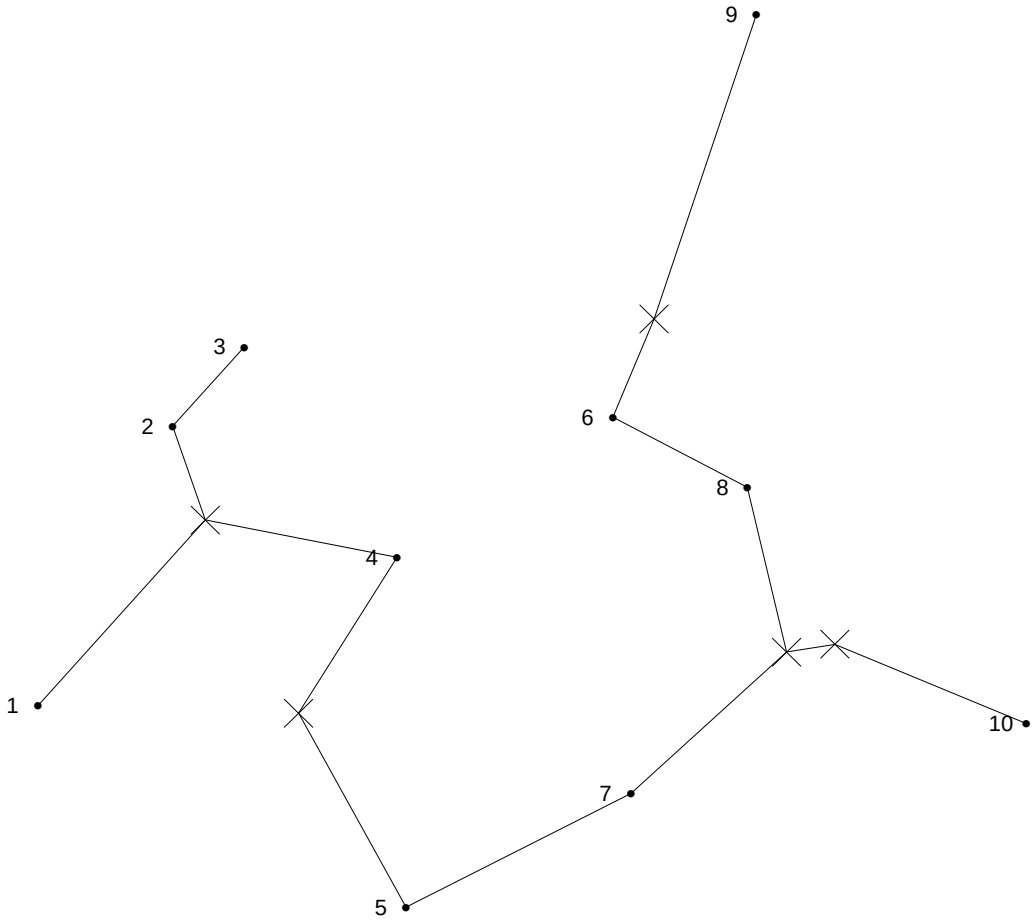


Figure 7.3. Point A in Figure 7.2 has been deleted. The minimum spanning tree has a slightly better "look" in the region of points 1, 2, 3 and 4, but there is still room for much improvement.

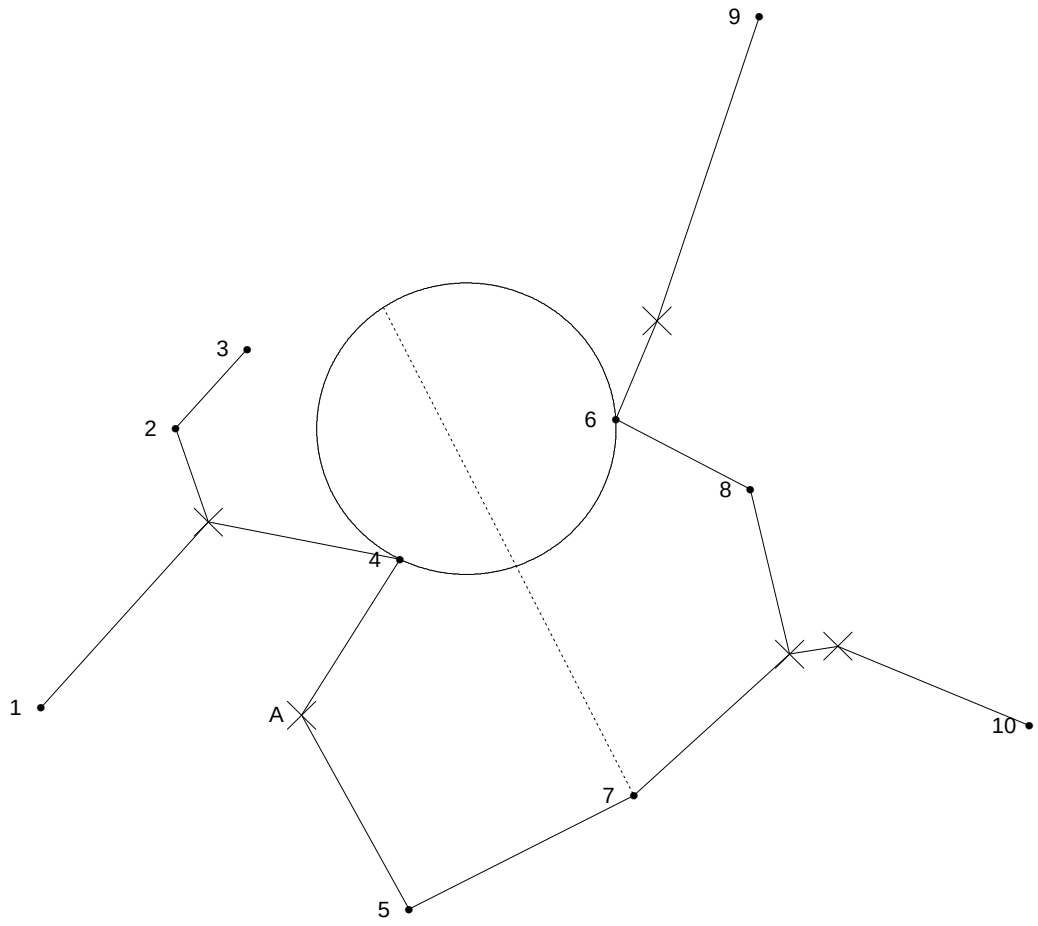


Figure 7.4. Point A is replaced by the point defined by the triangle formed by points 4, 6 and 8. The new point is the at the intersection of the 120° arc from 4 to 6 and the axis of the three point FST (dashed line).

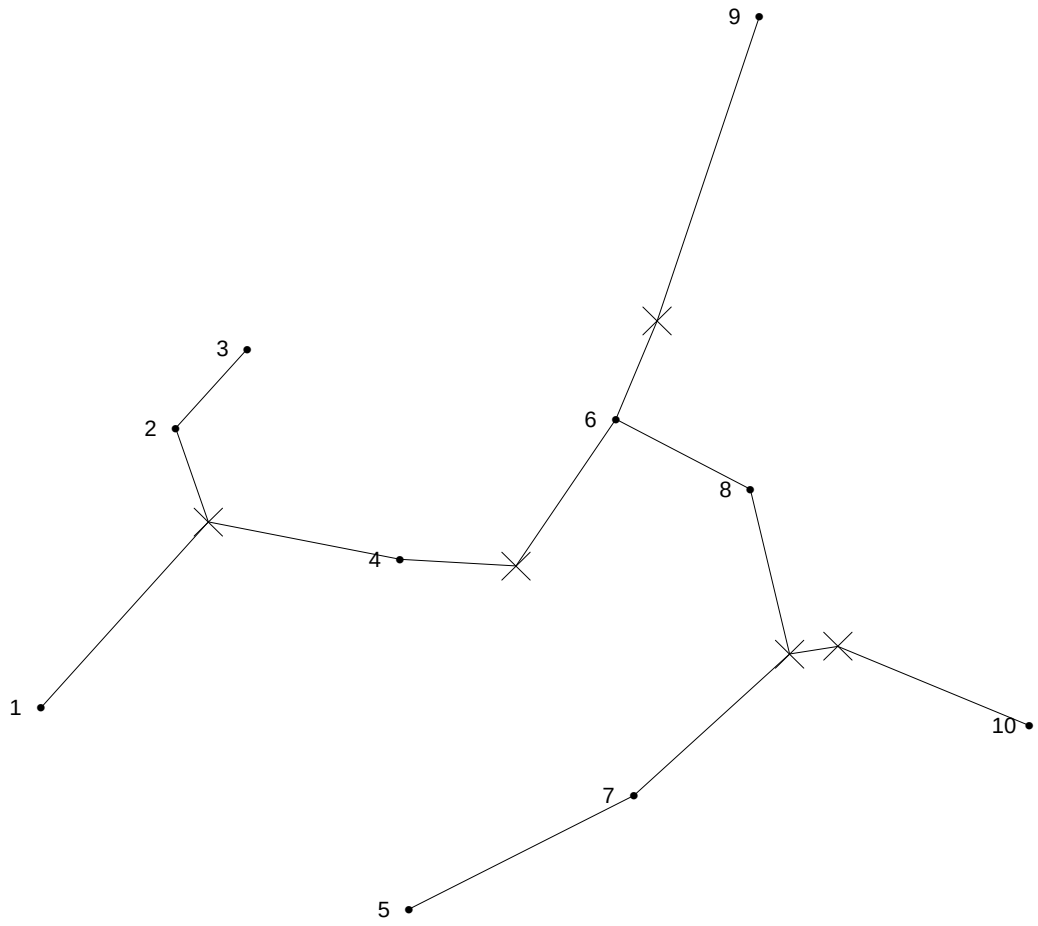


Figure 7.5. Point A in Figure 7.4 is replaced by a point based on 4, 6 and 7. The new point, to the right of point 4, is connected to 4 and 6 in the new minimum spanning tree but is not connected to point 7.

an optimal, or near optimal, topology, but is not quite in the correct location. It needs to be moved. If only deletion and addition changes are possible it is unlikely it will be deleted because removing it would give an increase in cost. The replace change does provide the opportunity for the point to be replaced or moved to its optimal location with respect to the topology.

In the early stages of the annealing when most increasing cost changes are accepted it is possible for a replace change to give a dramatic change to the minimum spanning tree. The replacement of A in Figure 7.4 with a point defined by 4, 6 and 7 to give the solution shown in Figure 7.5 is a good example of this.

7.2.4 Which change to use?

At each transition there must be some way of determining which change to apply. Clearly when $k = 0$ only an addition can be performed, and when $k = n - 2$ delete and replace are the only allowable changes. A simple method is to select a change type randomly with equal probability and allow for the restrictions imposed when there are no points or $n - 2$ points in the current solution. This approach is used in the simulated annealing algorithm presented in this chapter.

7.2.5 The neighbourhood size

Fundamental to the polynomial cooling schedule is the neighbourhood size. The maximum neighbourhood size over all possible solutions determines the number of transitions attempted at each level of the control parameter. This number is called the chain length (see Section 5.4.1). For the travelling salesman problem using a 2-change transition the maximum neighbourhood size is $O(n^2)$ and the number of possible solutions is $\frac{1}{2}(n-1)!$ for a symmetric problem (see Section 6.1.1).

In Section 2.6 an expression for the number of Steiner topologies for n points is shown, and is calculated for some n (see Table 2.2). The numbers are frighteningly large! The add, delete and replace changes do not necessarily give Steiner topologies. The Steiner points can have any degree in the minimum spanning tree, therefore the number of possible topologies is larger than the number of Steiner topologies. No attempt is made to count the number of topologies that can be obtained from the current solution by making a change, for example adding another point.

An alternative is to consider how many possible changes can be made. For instance if there are k points in the current solution, how many different points can be added? If it is assumed every combination of three points has a Steiner point then there are $\begin{pmatrix} n+k \\ 3 \end{pmatrix}$ points that can be added. This is $O(n^3)$. What if there are $k = n - 2$ points and one is replaced? One of the k must be deleted and another added by choosing three points from the remaining $n + n - 3$ points to form a new point. Again assuming every

combination of three has a Steiner point then the number of combinations is $(n-2) \times \begin{pmatrix} n+n-3 \\ 3 \end{pmatrix}$. This is $O(n^4)$ and is an upper bound on the maximum size of a neighbourhood. Unfortunately even for moderate size problems this is a large number, for example $50^4 = 6,250,000$ and $100^4 = 100,000,000$. For the travelling salesman problem the corresponding numbers are only 1,175 and 4,850. Using an $O(n^4)$ chain length is clearly unreasonable. An $O(n^2)$ length, although somewhat arbitrary, at least reduces the computation time and still gives sufficient numbers of transitions. A chain length of n^2 is used in the Euclidean Steiner tree problem simulated annealing implementation. However, the n^4 length is not completely abandoned. Some experiments are performed below to show it is excessive and impractical.

Figure 7.6 shows the evolution of a Steiner solution using simulated annealing. The Steiner points move to locations where they are degree three with all angles approximately equal to 120°. Figure 7.7 shows ten simulated annealing solutions for one of Cockayne and Hewgill's test problems. The simulated annealing program is listed and its implementation briefly discussed in Appendix H.

7.3 Empirical Results

The relatively simple simulated annealing algorithm described above is evaluated in this section. Firstly, performance on randomly generated problems of different sizes is analysed. This gives an indication of the time complexity of the algorithm. Secondly, Cockayne and Hewgill's test problems are used to compare simulated annealing with the results of Beasley and Goffinet [2]. The comparison is one of quality, not computation time.

7.3.1 Initial experiments

Results for both computation time and quality of solution are presented when using the simulated annealing implementation to solve a range of randomly generated problems varying in size from ten to one hundred points. Chain lengths of n^2 and n^4 are used. Further, using n^2 a *coarse* and *fine* set of polynomial schedule parameters are used. Coarse in the sense that solutions are found "quickly" without necessarily being "good" solutions and fine in the sense that more time is spent looking for good solutions. The two sets of parameters are shown in Table 7.1.

For problem sizes $n = 10, 20, \ldots, 90, 100, 150, 200$ the simulated annealing implementation is used to "solve" a randomly generated problem five times, each time with a different set of random number generator seeds. This is done using the coarse and fine parameter sets and a chain length of n^2. Further, a three hour CPU time limit is enforced.[3] In no cases can solutions to the 150 and

[3]The three hour limit is arbitrary, but was decided upon after considering the time necessary to accumulate experimental results and what felt reasonable for a heuristic for the Euclidean Steiner tree problem.

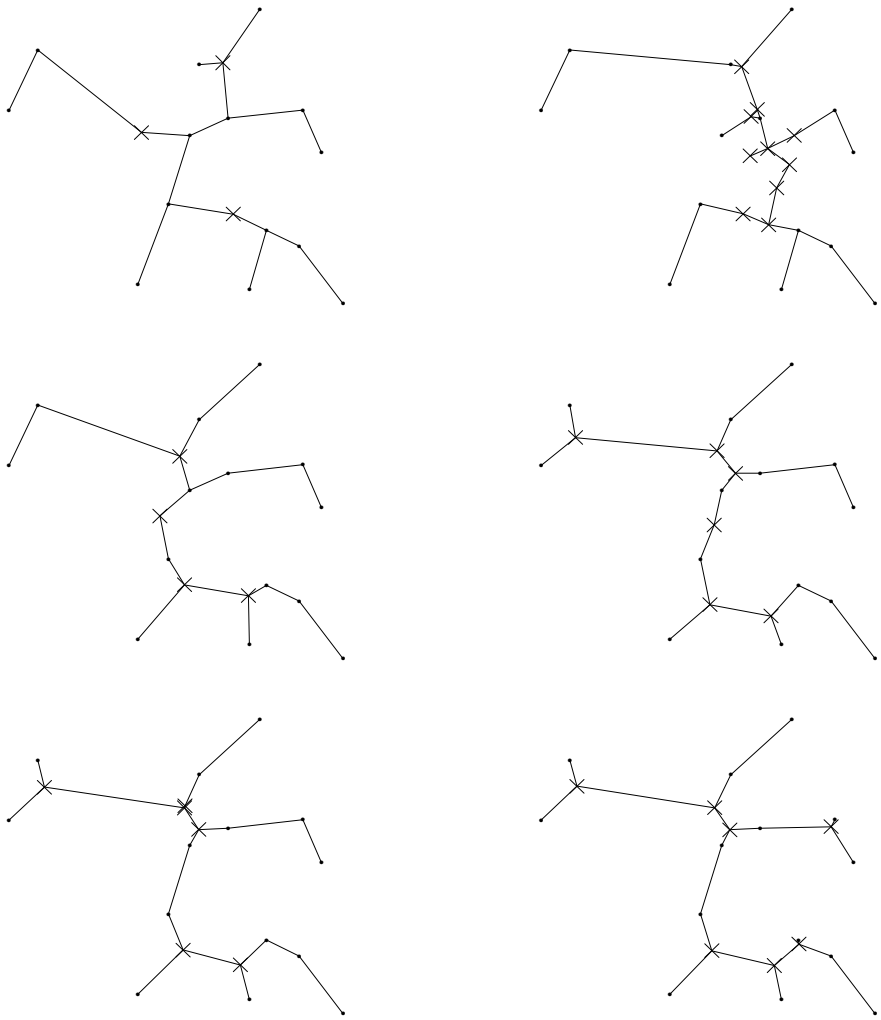


Figure 7.6. The six plots show the solution at the end of selected chains of a simulated annealing run for a fifteen point Euclidean Steiner tree problem. The crosses are the points in the solution set. *The order of the plots is left to right, top to bottom.*

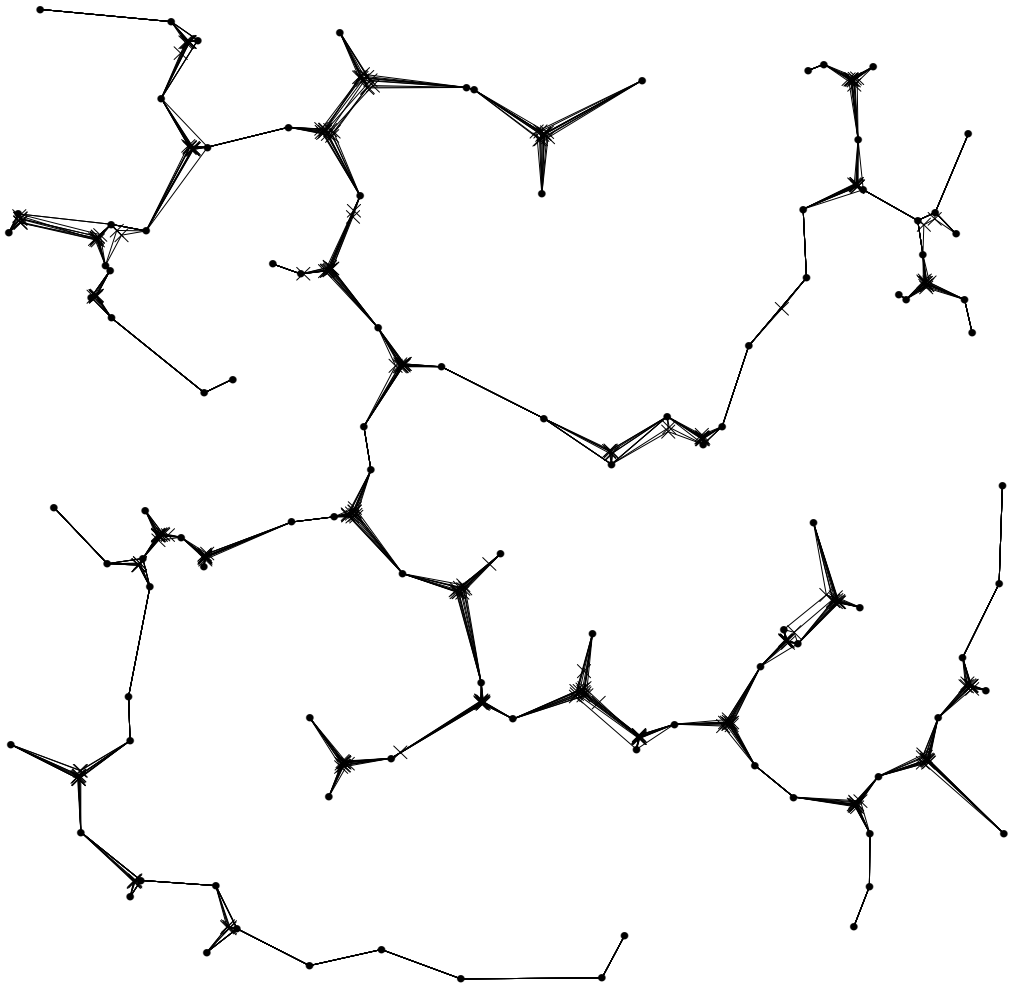


Figure 7.7. Ten simulated annealing solutions for Cockayne and Hewgill's Problem 3. The dots are the one hundred given points and the crosses are the Steiner points in the various solutions. The solutions are all very similar. It is the fine detail that remains to be resolved.

Parameter	Coarse	Fine
Distance Parameter δ	10	1
Initial Acceptance Ratio ϕ_0	0.9	0.99
Stopping Parameter ϵ_s	10^{-3}	10^{-5}

Table 7.1. The two sets of polynomial cooling schedule parameters used to investigate the application of simulated annealing to the Euclidean Steiner tree problem. The fine set is the base set of parameters used in Chapter 6 to study simulated annealing when applied to the travelling salesman problem.

200 point problems be found within three hours. In addition no 90 and 100 point problems finish within three hours using the fine set. Table 7.2 shows the mean computation times and *sample* standard deviations using the coarse and fine sets. As expected the coarse runs are faster. The computations times are also less variable than the fine set times.

Problem Size	*Coarse*		*Fine*	
n	Mean	Standard Deviation	Mean	Standard Deviation
10	2.44	0.20	5.07	0.59
20	19.21	1.04	59.70	4.66
30	83.52	21.67	314.24	49.63
40	254.47	80.62	752.90	55.63
50	359.42	47.18	1467.04	173.19
60	851.03	35.71	3573.98	1024.90
70	1351.67	118.63	5445.85	478.01
80	2234.71	166.33	8527.74	1637.63
90	2964.75	295.15	-	-
100	4606.40	261.42	-	-
150	-	-	-	-
200	-	-	-	-

Table 7.2. The mean computation times and *sample* standard deviations in seconds for the randomly generated problems. The samples are all of size five. For the larger problems no solutions were found within a three hour CPU time limit.

A simple least squares regression shows the coarse mean times to be $O(n^{3.29})$ and the fine mean times to be $O(n^{3.58})$. Clearly, simulated annealing is unable to compete with tailored heuristics. For example, Beasley and Goffinet's heuristic is $O(n^{2.19})$. For a 100 point problem their heuristic is about 100 times faster than simulated annealing (as implemented above).

Although much slower can coarse annealing give better quality solutions than Beasley and Goffinet? The answer to this question is postponed until the section below which analyses simulated annealing's performance using Cockayne and Hewgill's test problems. Instead at this point the difference in quality of solution between coarse and fine solutions is studied. If the difference is small then abandonment of the fine set of schedule parameters is not such a

dangerous thing to do in favour of the faster coarse set. Regardless, the fine set is of limited use because it is desired to investigate one hundred point problems below. One hundred point problems require more than three hours of computation time using the fine set.[4]

For each problem the mean, minimum and maximum cost of the five solutions are shown in Table 7.3. The cost is the length of the minimum spanning tree of the given points and solution points as a percentage of the length of the minimum spanning tree of the given points only. The differences between the solutions from coarse and fine solutions are marginal. For all problems the best fine solution is better than the best coarse solution. However the difference is small. The largest difference between minima of coarse and fine solutions is 0.19%: the 60 point problem minimum solution costs are 96.70% and 96.51% for coarse and fine respectively, a difference of 0.19%. Similar maximum differences for means and maxima are 0.25% and 0.45% respectively. The negligible difference between the quality of solutions produced by the two parameter sets is a direct result of the way simulated annealing works: the overall structure is determined then the fine detail is slowly identified. The coarse set quickly determines the approximate positions of the Steiner points. The fine set also does this and takes the process one step further and works on the very fine detail, that is small changes are made to the positions of already discovered Steiner points. Unfortunately the gain from this second stage is insignificant compared to the large increase in computation time.

Problem Size	*Coarse*			*Fine*		
n	Minimum	Mean	Maximum	Minimum	Mean	Maximum
10	96.19	96.27	96.49	96.19	96.20	96.22
20	95.64	95.76	95.86	95.57	95.70	95.90
30	96.08	96.21	96.42	95.95	95.96	95.97
40	96.99	97.16	97.28	96.98	97.03	97.12
50	96.59	96.70	96.97	96.51	96.52	96.55
60	96.70	96.76	96.81	96.51	96.56	96.64
70	95.70	95.80	95.94	95.63	95.64	95.65
80	96.19	96.23	96.33	96.13	96.17	96.31
90	96.78	96.83	96.94	-	-	-
100	96.57	96.60	96.62	-	-	-

Table 7.3. The minimum, mean and maximum solution costs of the random problems simulated annealing solutions for both the coarse and fine schedule parameter sets. The fine solutions are only marginally better than the coarse solutions.

A similar analysis of simulated annealing using a chain length of n^4 instead of n^2 with the coarse set of parameters indicates the computation time to be $O(n^{3.44})$. The quality of solutions is generally between that of the coarse and fine set using a chain length of n^2.

[4]The rough and ready regression indicates a time of five to six hours is likely using the fine set compared to about one to one and a half hours for the coarse set.

7.3.2 Comparison with Beasley and Goffinet's results

It is shown above that simulated annealing can not compete with the tailored heuristics for the Euclidean Steiner tree problem in terms of computation time. However, the quality of annealing solutions may be better. This is investigated in this section.

Using a chain length of n^2 and the coarse annealing parameter set, simulated annealing is applied ten times to each of Cockayne and Hewgill's thirty 100 point test problems.[5] The annealing results are compared to Beasley and Goffinet.[6] Table 7.4 shows the annealing results and Beasley and Goffinet's results. An initial browse of the table indicates there to be little difference between the two heuristics. Further, this is using the coarse annealing schedule parameters. Table 7.5 shows a simple comparison of the annealing and Beasley and Goffinet results. The table shows which heuristic is better for each problem assuming the Beasley and Goffinet values are either minima or means.

Simple statistics are shown in Table 7.6. The statistics show Beasley and Goffinet's to be the better heuristic. This heuristic is only bettered by simulated annealing in the worst percentage above optimal measure. Beasley and Goffinet's heuristic is clearly the winner assuming their results are means. But the distinction is less clear when assuming minima. The average percentages above optimal are 0.12% and 0.11% for simulated annealing and Beasley and Goffinet's heuristic respectively. The sign test does *not* reject the hypothesis that the two heuristics are of equal quality in favour of Beasley and Goffinet's being better.[7]

The empirical analysis suggests there is little difference, if any, between coarse simulated annealing and Beasley and Goffinet's heuristic, assuming their results are minima.

[5] Appendix B contains the given points and Steiner points of the optimal solutions to the test problems.

[6] The results are shown in Table 4 of Beasley and Goffinet [2]. It is not clear from the paper if the heuristic solution costs are means or minima from many runs of the heuristic on the test problems. The heuristic has a *small* stochastic element to avoid cycling, therefore strictly several runs are required on the same problem to obtain reliable data (see Section 4.3). There is no statistical information in the paper. Because of this Beasley and Goffinet's results are compared to both minima and means of simulated annealing results.

[7] Most statistics texts have a chapter on non-parametric statistics. Excluding the ties, simulated annealing was better 10 times and Beasley and Goffinet's heuristic 17 times. The probability of the latter being better than the former 17 or more times in 27 trials assuming they are identical is $0.5^{27} \sum_{i=17}^{27} {}^{27}\mathrm{C}_i = 0.06$. Small but perhaps not small enough to conclude they are different.

Problem	*Simulated Annealing*			*Beasley and Goffinet*
	Minimum	Mean	Maximum	
1	0.13	0.29	0.47	0.13
2	0.08	0.16	0.28	0.03
3	0.12	0.21	0.32	0.09
4	0.10	0.22	0.38	0.04
5	0.09	0.14	0.18	0.03
6	0.09	0.16	0.22	0.06
7	0.11	0.20	0.33	0.15
8	0.10	0.24	0.46	0.05
9	0.15	0.24	0.32	0.02
10	0.16	0.27	0.54	0.47
11	0.15	0.26	0.40	0.12
12	0.07	0.12	0.18	0.00
13	0.11	0.19	0.31	0.19
14	0.14	0.26	0.38	0.15
15	0.14	0.23	0.43	0.02
16	0.12	0.17	0.21	0.44
17	0.11	0.17	0.24	0.00
18	0.15	0.20	0.28	0.01
19	0.12	0.16	0.20	0.08
20	0.10	0.18	0.37	0.13
21	0.08	0.21	0.30	0.08
22	0.10	0.20	0.26	0.10
23	0.18	0.26	0.44	0.02
24	0.15	0.20	0.24	0.16
25	0.12	0.25	0.39	0.02
26	0.12	0.18	0.24	0.20
27	0.17	0.27	0.34	0.47
28	0.09	0.13	0.23	0.00
29	0.14	0.20	0.39	0.07
30	0.09	0.14	0.21	0.11
All	0.07	0.20	0.54	0.11

Table 7.4. The simulated annealing results from ten runs on each of Cockayne and Hewgill's one hundred point test problems. The minimum, mean and maximum of solution costs as percentages above optimal from annealing are shown for each problem. The Beasley and Goffinet solution costs are from [2]. For Problems 17 and 28 their heuristic found the optimal solution. Problem 12 has a percentage above optimal value of 0.00. This is due to rounding. The optimal solution length and Beasley and Goffinet's solution length differ in the fifth decimal place.

Problem	*Assuming a Minimum*		*Assuming a Mean*	
	Simulated Annealing	Beasley and Goffinet	Simulated Annealing	Beasley and Goffinet
1	●	●		●
2		●		●
3		●		●
4		●		●
5		●		●
6		●		●
7	●			●
8		●		●
9		●		●
10	●		●	
11		●		●
12		●		●
13	●		●	●
14	●			●
15		●		●
16	●		●	
17		●		●
18		●		●
19		●		●
20	●			●
21	●	●		●
22	●	●		●
23		●		●
24	●			●
25		●		●
26	●		●	
27	●		●	
28		●		●
29		●		●
30	●			●

Table 7.5. A dot indicates which heuristic is better. The comparison is done assuming the Beasley and Goffinet value is both a minimum and a mean and therefore is compared to the simulated annealing minimum and mean respectively. For some problem instances the values are the same. In these cases dots are placed in both columns, for example Problem 1 assuming a minimum.

	Assuming a Minimum		*Assuming a Mean*	
	Simulated Annealing	Beasley and Goffinet	Simulated Annealing	Beasley and Goffinet
Number of times first or tied for first	13	**20**	5	**26**
Average percentage above the optimal solution	0.12%	**0.11%**	0.20%	**0.11%**
Average rank	1.62	**1.38**	1.85	**1.15**
Worst percentage above optimal solution	**0.18%**	0.47%	**0.29%**	0.47%

Table 7.6. Statistics for comparing the two heuristics. The *Assuming a Minimum* Simulated Annealing column uses the minima results of simulated annealing for comparison with Beasley and Goffinet's results. The *Assuming a Mean* Simulated Annealing column uses the means results of simulated annealing. In all but one category Beasley and Goffinet's heuristic is better then simulated annealing for both assumptions.

7.4 Local Improvement

The simulated annealing solutions are on average 0.20% above optimal. This is a small margin. Regardless there is room for improvement, especially as Beasley and Goffinet's equivalent margin is 0.11%. In this section a very simple local improvement procedure is applied to the simulated annealing solutions.

A more complicated but perhaps more rewarding improvement mechanism is to use the simulated annealing solution as the starting point for Beasley and Goffinet's heuristic. The heuristic normally begins with an empty set of "Steiner points" with the best solution equal to the minimum spanning tree of the given points. It would be hoped that the heuristic preserves the underlying topology discovered by the annealing and quickly finds the optimal locations of the Steiner points.

7.4.1 The simple procedure

The local improvement procedure assumes the solution contains points that are close to their optimal location and are degree three. This assumption means it is highly likely that the Steiner point defined by a solution point's three neighbours does exist.

Although contrary to the assumption the procedure first of all checks for points of degree one and two and removes them if found. The procedure performs the following step iteratively until the difference in length of the solutions at the beginning and end of an iteration as a percentage of the minimum spanning tree of the given points is less than some small number, for example 0.001%. An iteration is a single pass through the set of solution points, where each point is replaced by the Steiner point defined by its three neighbours, but only if this gives a reduction in the solution cost after removing degree one and two points that may be created by the change. During an iteration it is possible for the size of the solution set to decrease, it can never increase. Because only a single pass is made a point that has already been processed and is a neighbour of a point or points that has or have also changed may no longer be in its optimal position with respect to its neighbours. Such a point is not re-processed in the iteration.

The procedure is straightforward. No attempts are made to eliminate angles of less than 120° through the introduction of more Steiner points. However, only decreasing cost changes are accepted.

7.4.2 Local improvement results

The simulated annealing with local improvement solution costs as a percentage of optimal are shown in Table 7.7 together with Beasley and Goffinet's values. The average percentage above optimal over all problems has decreased by 0.09% from 0.20% to give 0.11%. The reductions achieved are shown in Table 7.8. The smallest reduction was 0.03% and the largest 0.23%.

Tables 7.9 and 7.10 show the simple comparison of the heuristics and

Problem	*Simulated Annealing*			*Beasley and Goffinet*
	Minimum	Mean	Maximum	
1	0.03	0.19	0.38	0.13
2	0.02	0.09	0.19	0.03
3	0.07	0.14	0.25	0.09
4	0.05	0.13	0.24	0.04
5	0.04	0.07	0.12	0.03
6	0.04	0.09	0.16	0.06
7	0.05	0.11	0.21	0.15
8	0.03	0.13	0.32	0.05
9	0.08	0.16	0.22	0.02
10	0.08	0.18	0.40	0.47
11	0.08	0.16	0.31	0.12
12	0.01	0.04	0.06	0.00
13	0.02	0.10	0.23	0.19
14	0.07	0.18	0.29	0.15
15	0.03	0.13	0.31	0.02
16	0.05	0.08	0.12	0.44
17	0.03	0.07	0.09	0.00
18	0.04	0.09	0.14	0.01
19	0.06	0.08	0.13	0.08
20	0.04	0.11	0.25	0.13
21	0.03	0.10	0.19	0.08
22	0.02	0.10	0.17	0.10
23	0.05	0.14	0.21	0.02
24	0.06	0.11	0.16	0.16
25	0.06	0.14	0.25	0.02
26	0.05	0.09	0.12	0.20
27	0.11	0.19	0.25	0.47
28	0.02	0.05	0.12	0.00
29	0.06	0.10	0.22	0.07
30	0.02	0.06	0.12	0.11
All	0.01	0.11	0.40	0.11

Table 7.7. The simulated annealing *with local improvement* results. The values are percentages above optimal.

Problem	*Reduction as a percentage of optimal*		
	Minimum	Mean	Maximum
1	0.06	0.09	0.12
2	0.04	0.07	0.12
3	0.04	0.07	0.09
4	0.05	0.09	0.14
5	0.05	0.07	0.10
6	0.05	0.07	0.09
7	0.06	0.09	0.15
8	0.07	0.11	0.14
9	0.05	0.08	0.11
10	0.05	0.09	0.14
11	0.07	0.09	0.13
12	0.05	0.08	0.13
13	0.03	0.09	0.15
14	0.06	0.08	0.10
15	0.05	0.10	0.19
16	0.05	0.09	0.14
17	0.07	0.11	0.14
18	0.06	0.11	0.16
19	0.04	0.08	0.12
20	0.05	0.07	0.12
21	0.05	0.10	0.20
22	0.07	0.10	0.16
23	0.05	0.12	0.23
24	0.05	0.09	0.12
25	0.05	0.11	0.15
26	0.07	0.09	0.12
27	0.06	0.08	0.12
28	0.06	0.09	0.15
29	0.05	0.10	0.17
30	0.06	0.08	0.10
All	0.03	0.09	0.23

Table 7.8. The reductions as percentages of optimal achieved by applying local improvement to simulated annealing solutions. As many of the simulated annealing solutions are close to optimal the gain from local improvement is small. The average reduction over all problems is only 0.09%.

summary statistics respectively. Simulated annealing dominates Beasley and Goffinet's heuristic assuming their results are minima. The average percentage above optimal for simulated annealing with local improvement is 0.05% compared to Beasley and Goffinet's 0.11%. Annealing is ranked first two-thirds of the time. The probability of being ranked first 20 or more times in 30 trials assuming the heuristics are identical is only 0.02.

Beasley and Goffinet's heuristic is still ahead of simulated annealing assuming their results to be means. However the lead is not as great as without local improvement. The average percentages are both 0.11%, and annealing has a lower worst percentage. Although much closer annealing is second 9 times out of 28 (the 2 ties are not counted).

7.5 Summary

The above analyses of simulated annealing with and without local improvement indicate very good quality solutions for the Euclidean Steiner tree problem can be found using annealing. Further, the solutions compare favourably with those of the best known tailored heuristic for the problem. Both the annealing and local improvement used are simple minded approaches. More sophisticated methods may exist. In particular, methods that improve the computation time aspect of annealing, whether it be through modifications to the annealing schedule or the transition mechanisms employed.

Importantly, it has been demonstrated a simple annealing approach does work for the problem. To extend annealing to solve generalisations or focus it to tackle specialisations of the Euclidean Steiner tree problem is now a possibility.[8] Although non-trivial tasks, they are likely to be less intensive than constructing tailored heuristics. This is one advantage of simulated annealing over tailored heuristics.

The simulated annealing approach developed for the Euclidean Steiner tree problem can be modified to find good solutions to problems such as augmenting an existing telecommunications network or finding a minimum cost network. Problems less abstract than the underlying Euclidean Steiner tree problem, but capable of being applied to real situations.

[8]Chapter 3 looks at three applications of the Euclidean Steiner tree problem. All are generalisations and therefore more difficult to solve.

Problem	*Assuming a Minimum*		*Assuming a Mean*	
	Simulated Annealing	Beasley and Goffinet	Simulated Annealing	Beasley and Goffinet
1	●			●
2	●			●
3	●			●
4		●		●
5		●		●
6	●			●
7	●		●	
8	●			●
9		●		●
10	●		●	
11	●			●
12		●		●
13	●		●	
14	●			●
15		●		●
16	●		●	
17		●		●
18		●		●
19	●		●	●
20	●		●	
21	●			●
22	●		●	●
23		●		●
24	●		●	
25		●		●
26	●		●	
27	●		●	
28		●		●
29	●			●
30	●		●	

Table 7.9. The comparison of simulated annealing *with local improvement* and Beasley and Goffinet's heuristic.

	Assuming a Minimum		*Assuming a Mean*	
	Simulated Annealing	Beasley and Goffinet	Simulated Annealing	Beasley and Goffinet
Number of times first or tied for first	**20**	10	11	**21**
Average percentage above the optimal solution	**0.05%**	0.11%	**0.11%**	**0.11%**
Average rank	**1.33**	1.67	1.67	**1.33**
Worst percentage above optimal solution	**0.11%**	0.47%	**0.19%**	0.47%

Table 7.10. Summary statistics of the simulated annealing *with local improvement* results.

Appendix A

Listing of the Steiner Polygon Program

This appendix is a listing of the C++ program to compute the Steiner polygon. The LEDA library[30] is used with GNU g++ 2.2.3 on a SPARCstation running SunOS 4.1.3 with OpenWindows 3.0 and X11 Release 5. One particular machine is used for running programs: a SPARCstation 10/512 (dual 50MHz processor) with 160MB main memory and 400MB swap space. The program is run either directly from the Unix prompt or from S-PLUS 3.1.

The file containing **main()** is SMT.cc. The various files making up the program are listed in alphabetical order, except the make file called Makefile is listed first.

The necessary inputs to the program are the number of points, an integer giving the desired level of output, and if the output level equals zero then a list of the point coordinates. An output level of zero is primarily used when running the program from S-PLUS.

The output of the program depends on the output level input parameter. The higher the output level the more detailed the output. Table A.1 shows the output produced for each level of the output level parameter.

Output Level	Description
$= 0$	The number of points, size of the Steiner polygon and the number of points appearing more once on the Steiner polygon are output. If this last number is greater than zero then the Steiner polygon is degenerate.
≥ 3	The coordinates of the points and the Steiner polygon are output and if the Steiner polygon is degenerate the points at which it is degenerate are output.
≥ 11	As for 3 and the details of the construction of the Steiner polygon are output.

Table A.1. A description of the output produced by the Steiner polygon program.

A.1 Makefile

```
PROGRAM = SteinerPolygon
SOURCES = SMT STEINER_POLYGON ANGLE

LEDA = /u/grads/geoff/LEDA
X11 = /usr/openwin/lib

OBJECTS = $(SOURCES:%=%.o)

SUFFIXES = .o .c .c~ .cc .cc~ .s .s~ .S .S~ .ln .f .f~ .F .F~ .l .l~ \
        .mod .mod~ .sym .def .def~ .p .p~ .r .r~ .y .y~ .h .h~ .sh .sh~ \
        .cps .cps~

.SUFFIXES: $(SUFFIXES)
.SILENT:
.KEEP_STATE:

CCC=g++
CCFLAGS=-O2 -I$(LEDA)/incl
CPPFLAGS=
LDFLAGS=
LEDALIBS=-lP -lG -lL
#X11LIBS=-lWx -lxview -lolgx -lX11
X11LIBS=

COMPILE.cc=$(CCC) $(CCFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
LINK.cc=$(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH)

$(PROGRAM): $(OBJECTS)
        @echo --- Linking to create $(PROGRAM)
        $(LINK.cc) -o $(PROGRAM) $(OBJECTS) -L$(LEDA) -L$(X11) $(LEDALIBS)
$(X11LIBS) -lm
.cc.o:
        @echo --- Compiling $*
        $(COMPILE.cc) $<

clean:
        @echo --- Removing all intermediate files and the executable file
        rm -f *.o *.cc~ core a.out $(PROGRAM) Makefile~ *.h~ mon.out *~
```

A.2 ANGLE.h

```
#ifndef ANGLE_H
#define ANGLE_H

#include <LEDA/plane.h>

double ANGLE(const point&, const point&, const point&);

#endif
```

A.3 ANGLE.cc

```
#include "ANGLE.h"

double ANGLE(const point& q, const point& p, const point& r){

  // Finds the angle between two line segments qp and pr
  // The angle is always less than 180 degrees

  point s = r;

  double omega = segment(p, q).direction();
  s = s.rotate(p, -omega);
  double theta = segment(p, s).direction();
  if (theta<0) theta *= -1;

  return theta;

}
```

A.4 IntegerSet.h

```
#ifndef INTEGERSET_H
#define INTEGERSET_H

#include <LEDA/basic.h>

class int_set {

  // This is basically the LEDA int_set class with minor changes and additions

  char* V;
  int size;
  int low;
  char mask[8];

public:

  int_set(){};
  int_set(int n);
  int_set(int,int);
  int_set(const int_set&);
  ~int_set() { delete V; }

  void clear();
  void insert(int);
  void del(int);

  int member(int) const;

  int_set& join(const int_set&);
  int_set& intersect(const int_set&);
  int_set& complement();

  int_set& operator=(const int_set&);
```

```
  int_set operator|(const int_set&);
  int_set operator&(const int_set&);
  int_set operator~();

  void print(ostream& out = cout) const{

    int i;

    cout ≪ "{";
    loop(i, low, low+size-1) if (member(i)) out ≪ string("%3d", i);
    cout ≪ "  }";

  }

friend void Print(const int_set&, ostream& = cout);
friend void Read(int_set&, istream& = cin);
friend int Size(const int_set&);
friend int compare(const int_set&, const int_set&);

};

inline void Print(const int_set& S, ostream& out){

  int i;

  cout ≪ "{";
  loop(i, S.low, S.low+S.size-1) if (S.member(i)) out ≪ string("%3d", i);
  cout ≪ "  }";

}

inline void Read(int_set& S, istream& in){};

inline int Size(const int_set& S){

  int n = 0;
  int i;

  loop(i, S.low, S.low+S.size-1) if (S.member(i)) ++n;

  return n;

}

inline int compare(const int_set& S, const int_set& T){ return 0; }

#endif
```

A.5 PointArray.h

```
#ifndef POINTARRAY_H
#define POINTARRAY_H
```

```
#include <LEDA/plane.h>
#include <LEDA/basic.h>
#include <LEDA/array.h>

class PointArray : public array<point>{

private:

  int size;

public:

  PointArray(int s) : array<point>(1, Max(s, 1)), size(s){};
  ~PointArray(){};

  void Number(int s){
    if((s<1) || (s>high()))
      error_handler(1, "PointArray:  number of points inappropriate");
    else size = s;
  }

  void Print(){
    int i;
    loop(i, 1, size) cout << (*this)[i] << "  ";
  }

};

#endif
```

A.6 SMT.h

```
#ifndef SMT_H
#define SMT_H

#include <LEDA/basic.h>
#include <LEDA/array.h>
#include <LEDA/list.h>
#include <LEDA/plane.h>
#include <LEDA/ugraph.h>
#include <LEDA/dictionary.h>

typedef array<int> IntegerArray;
typedef list<point> PointList;
typedef dictionary<point, int> PointIntegerDictionary;
typedef ugraph Topology;
typedef Topology* TopologyPtr;
typedef node_array<point> NodePointArray;
typedef array<node> NodeArray;
typedef NodeArray* NodeArrayPtr;

extern PointIntegerDictionary PointIntegerD;
extern float maxTimeSeconds;
extern int OutputLevel;
extern int numPoints;
```

```
#include "PointArray.h"
#include "SteinerTree.h"
#include "IntegerSet.h"

typedef int_set IntegerSet;
typedef IntegerSet* IntegerSetPtr;
typedef array<IntegerSetPtr> IntegerSetArray;

extern PointArray Point;

const double _120 = 2*PI/3;
const double _60 = PI/3;

#endif
```

A.7 SMT.cc

```
const int maxPoints = 100;

#include "SMT.h"
#include "STEINER_POLYGON.h"
#include "ANGLE.h"

PointArray Point(maxPoints);
PointIntegerDictionary PointIntegerD;

int OutputLevel;
int numPoints;

main()
{

  numPoints = read_int("");

  if (numPoints>maxPoints){
    cout << string("SMT: maximum is%3d\n", maxPoints);
    exit(1);
  }

  OutputLevel = read_int("");

  // Create the global array of points and the global dictionary indexed by
  // a point and giving its identification number or name

  int i;
  point p;

  if (OutputLevel==0){
    loop(i, 1, numPoints){
      Read(p);
      Point[i] = p;
      PointIntegerD.insert(p, i);
    }
  }
```

```
  if (OutputLevel≥1){
    init_random();
    double radius;
    double angle;
    loop(i, 1, numPoints){
//      radius = random(0,10000)/200;
//      angle = PI∗random(0,719)/360;
//      Point[i] = point(int(radius∗cos(angle)) + 50, int(radius∗sin(angle)) + 50);
      Point[i] = point(random(1,100),random(1,100));
      PointIntegerD.insert(Point[i], i);
    }
  }

  if (OutputLevel≥1){
    cout ≪ "SMT: Points\n";
    loop(i, 1, numPoints){
      cout ≪ string("SMT: [%3d] ", i) ≪ Point[i];
      newline;
    }
  }

  // Get the Steiner Polygon for the points

  PointList SteinerPolygon;
  STEINER_POLYGON(SteinerPolygon, PointList());

  // Check if the Steiner Polygon is not degenerate. If it is indicate where
  // and stop

  IntegerSet SteinerPolygonPoints(1, numPoints);
  IntegerSet DegeneratePoints(1, numPoints);
  bool Degenerate = false;

  forall(p, SteinerPolygon){
    i = PointIntegerD.access(p);
    if (SteinerPolygonPoints.member(i)){
      Degenerate = true;
      DegeneratePoints.insert(i);
    }
    SteinerPolygonPoints.insert(i);
  }

  if (Degenerate){
    if ((OutputLevel≥1) && (OutputLevel<3)){
      cout ≪ "SMT: Steiner Polygon\nSMT: ";
      forall(p, SteinerPolygon) cout ≪ string("%3d", PointIntegerD.access(p));
      newline;
    }
    if (OutputLevel≥1){
      cout ≪ "SMT: Steiner Polygon is degenerate at ";
      DegeneratePoints.print();
      newline;
    }
  }

  if (OutputLevel==0){
```

```
    cout ≪ string("%4d%4d%4d\n", numPoints, SteinerPolygon.length(),
Size(DegeneratePoints));
  }

//   if (OutputLevel>=1){
//     cout << "P$x_c(";
//     loop(i, 1, numPoints-1) cout << Point[i].xcoord() << ",";
//     cout << Point[numPoints].xcoord() << ")\n";
//     cout << "P$y_c(";
//     loop(i, 1, numPoints-1) cout << Point[i].ycoord() << ",";
//     cout << Point[numPoints].ycoord() << ")\n";
//   }

  return 0;

}
```

A.8 STEINER_POLYGON.h

```
#ifndef STEINER_POLYGON_H
#define STEINER_POLYGON_H

#include "SMT.h"

void STEINER_POLYGON(PointList&, PointList&);

#endif
```

A.9 STEINER_POLYGON.cc

```
#include "STEINER_POLYGON.h"

#include <LEDA/plane_alg.h>
#include "ANGLE.h"

void STEINER_POLYGON(PointList& SteinerPolygon, PointList& Points){

  bool Output = OutputLevel≥11;
  string Function = "STEINER_POLYGON: ";

  int i;
  bool empty;
  point p;

  // If the parameter Points is the empty list then the the Steiner Polygon
  // for all points is wanted

  empty = Points.empty();
  if (empty) loop(i, 1, numPoints) Points.append(Point[i]);

  // Get the convex hull using the LEDA algorithm

  PointList CH;
  CONVEX_HULL(CH, Points);
```

```
// Use the convex hull to initialise the Steiner Polygon

SteinerPolygon.clear();
while (!CH.empty()) SteinerPolygon.append(CH.Pop());

if (SteinerPolygon.size()<Points.size()){

  // Starting with the convex hull generate the Steiner Polygon

  PointList Possible;
  list_item LastChange, Current;
  bool Change, Empty;
  point q, r, s, t;
  double Angle, omega;
  segment Sqs;

  Current = SteinerPolygon.first();

  do{

    Change = false;

    // Get points of current and immediately succeeding position on the
    // Steiner Polygon

    q = SteinerPolygon.inf(Current);
    p = SteinerPolygon.inf(SteinerPolygon.cyclic_succ(Current));

    // Get a list of possible points for inclusion on the Steiner Polygon

    Possible.clear();
    forall(r, Points){
      if ((r≠q) && (r≠p)){
        if (ANGLE(p, q, r)<_60){
          Possible.append(r);
        }
      }
    }

    if (Output){
      cout ≪ Function ≪ "Steiner Polygon\n" ≪ Function;
      forall(r, SteinerPolygon) cout ≪ string("%3d", PointIntegerD.access(r));
      newline;
      cout ≪ Function ≪ string("Current %3d", PointIntegerD.access(q));
      cout ≪ string(" Next %3d\n", PointIntegerD.access(p));
      cout ≪ Function ≪ "Possibilities:";
      if (Possible.empty()) cout ≪ " none";
      else{
        forall(r, Possible)
          cout ≪ string("%3d", PointIntegerD.access(r));
      }
      newline;
    }


    // Consider each possible point in turn for inclusion on the
```

```
    // Steiner Polygon

    while ((!Possible.empty()) && (!Change)){

      r = Possible.pop();

      if (ANGLE(q, r, p)>_120){

        Angle = ANGLE(p, q, r);

        // Check if any point lies in the triangle with corners p, q, r

        Empty = true;

        forall(s, Points){

          // Don't use ANGLE to calculate the angle between segments pq and
          // qs because need to worry about sign

          omega = segment(q, p).direction();
          t = s.rotate(q, -omega);
          omega = -segment(q, t).direction();

          if ((omega>0) && (omega≤Angle)){
            if ((s≠r) && (s≠p) && (s≠q)){
              Sqs = segment(q, s);
              Empty = Empty && (Sqs.intersection(segment(p, r), t));
            }
          }
        }

        // Add point r to the Steiner Polygon if there are no points inside
        // the triangle

        if (Empty){
          SteinerPolygon.insert(r, Current, after);
          Change = true;
          if (Output)
            cout ≪ Function ≪ string("%3d empty\n", PointIntegerD.access(r));
        }
        else
          if (Output)
            cout ≪ Function ≪ string("%3d not empty\n", PointIntegerD.access(r));
      }
      else{
        if (Output)
            cout ≪ Function ≪ string("%3d angle less than 120\n",
PointIntegerD.access(r));
      }

    }

    if (!Change) Current = SteinerPolygon.cyclic_succ(Current);

  } while (Change || (Current≠SteinerPolygon.first()));

}
```

```
  if ((OutputLevel≥7) || (empty && (OutputLevel≥3))){
    cout ≪ "STEINER_POLYGON: Steiner Polygon\nSTEINER_POLYGON: ";
    forall(p, SteinerPolygon) cout ≪ string("%3d", PointIntegerD.access(p));
    newline;
  }

}
```

Appendix B

Cockayne and Hewgill's Test Problems

This appendix contains some of the thirty one hundred point Euclidean Steiner tree test problems solved to optimality by Cockayne and Hewgill [7]. The problems were kindly provided by Professor Ernest Cockayne and he has given permission for the data and solutions to be reproduced here.[1]

Only the problems used in the experiments with heuristic methods are shown in this appendix. The identification numbers of problems corresponds to Cockayne and Hewgill's original numbering of the problems. For each test problem the following are shown:

- The coordinates of the one hundred points.
- The length of the minimal spanning tree, Steiner minimal tree and the percentage reduction.
- The coordinates of the Steiner points in the Steiner minimal tree;
- A table showing the number of full Steiner trees of different sizes making up the SMT and the number of FSTs as percentages of the total number of full Steiner trees;
- A picture of the minimum spanning tree (dashed line) and Steiner minimal tree (solid line).

[1]Professor E J Cockayne, Department of Mathematics, University of Victoria, PO Box 1700, Victoria, British Columbia, CANADA V8W 2Y2. His e-mail address is DJMEC@UVVM.UVic.CA.

B.1 Test Problem 1

Minimum Spanning Tree	6.4487
Steiner Minimal Tree	6.2555
Reduction	3.00%

Given Points

1	(0.41010001, 0.00510000)	51	(0.35330001, 0.22270000)
2	(0.33080000, 0.02870000)	52	(0.16740000, 0.69770002)
3	(0.16390000, 0.05290000)	53	(0.86479998, 0.45179999)
4	(0.13200000, 0.01300000)	54	(0.36669999, 0.52689999)
5	(0.00000000, 0.21220000)	55	(0.24270000, 0.59020001)
6	(0.01640000, 0.24590001)	56	(0.66530001, 0.45320001)
7	(0.00980000, 0.34490001)	57	(0.29629999, 0.63779998)
8	(0.04230000, 0.44119999)	58	(0.67510003, 0.37549999)
9	(0.00330000, 0.49370000)	59	(0.18160000, 0.84130001)
10	(0.09300000, 0.78979999)	60	(0.40540001, 0.33939999)
11	(0.03550000, 0.92110002)	61	(0.33829999, 0.43349999)
12	(0.06240000, 0.93080002)	62	(0.23469999, 0.18070000)
13	(0.14080000, 0.99800003)	63	(0.09030000, 0.36370000)
14	(0.27930000, 0.97329998)	64	(0.88300002, 0.56720001)
15	(0.43340001, 0.93480003)	65	(0.53939998, 0.21230000)
16	(0.67439997, 0.87570000)	66	(0.32960001, 0.11930000)
17	(0.79540002, 0.95480001)	67	(0.29310000, 0.55409998)
18	(0.82590002, 0.94239998)	68	(0.93190002, 0.74540001)
19	(0.92729998, 0.99110001)	69	(0.51109999, 0.72140002)
20	(0.95429999, 0.98570001)	70	(0.35830000, 0.71780002)
21	(0.98960000, 0.76730001)	71	(0.61890000, 0.59090000)
22	(0.94889998, 0.64819998)	72	(0.82550001, 0.42690000)
23	(0.90280002, 0.57510000)	73	(0.34950000, 0.49380001)
24	(0.90469998, 0.56629997)	74	(0.19820000, 0.76289999)
25	(0.88700002, 0.53210002)	75	(0.74550003, 0.60939997)
26	(0.94270003, 0.40590000)	76	(0.83209997, 0.82770002)
27	(0.86559999, 0.37360001)	77	(0.08330000, 0.45159999)
28	(0.82980001, 0.27570000)	78	(0.43810001, 0.85409999)
29	(0.82929999, 0.24130000)	79	(0.57630002, 0.23750000)
30	(0.82630002, 0.10580000)	80	(0.11760000, 0.38409999)
31	(0.82220000, 0.04390000)	81	(0.88080001, 0.57380003)
32	(0.73070002, 0.06050000)	82	(0.07350000, 0.41000000)
33	(0.47830001, 0.03850000)	83	(0.36570001, 0.44610000)
34	(0.57359999, 0.82029998)	84	(0.22990000, 0.25260001)
35	(0.35479999, 0.45559999)	85	(0.24609999, 0.14320000)
36	(0.52370000, 0.30770001)	86	(0.12540001, 0.23160000)
37	(0.06510000, 0.26199999)	87	(0.53030002, 0.28479999)
38	(0.78369999, 0.50360000)	88	(0.81209999, 0.88919997)
39	(0.90039998, 0.69569999)	89	(0.64330000, 0.35350001)
40	(0.58679998, 0.47459999)	90	(0.47350001, 0.28670001)
41	(0.88789999, 0.57840002)	91	(0.26220000, 0.38450000)
42	(0.11920000, 0.93570000)	92	(0.01890000, 0.49360001)
43	(0.73189998, 0.51859999)	93	(0.59130001, 0.60890001)
44	(0.43939999, 0.59079999)	94	(0.90630001, 0.63169998)
45	(0.55559999, 0.45519999)	95	(0.71109998, 0.20330000)
46	(0.85240000, 0.82770002)	96	(0.13869999, 0.91360003)
47	(0.13740000, 0.91880000)	97	(0.67199999, 0.40400001)
48	(0.66750002, 0.13570000)	98	(0.13850001, 0.87150002)
49	(0.42710000, 0.55470002)	99	(0.51349998, 0.63779998)
50	(0.72420001, 0.22890000)	100	(0.91740000, 0.71640003)

Optimal Steiner Points

1	(0.01811600, 0.49246973)	22	(0.66435581, 0.47573689)
2	(0.06670050, 0.43481049)	23	(0.62281489, 0.49745488)
3	(0.09694912, 0.38088533)	24	(0.74509847, 0.52613550)
4	(0.04305354, 0.32983685)	25	(0.51876032, 0.79574901)
5	(0.04902345, 0.26928234)	26	(0.51893979, 0.64169705)
6	(0.21223302, 0.22605994)	27	(0.12823555, 0.93550313)
7	(0.55538243, 0.23813596)	28	(0.15284073, 0.97605461)
8	(0.67463833, 0.37571213)	29	(0.29384232, 0.67461854)
9	(0.81745493, 0.93867135)	30	(0.20104748, 0.72024119)
10	(0.83444732, 0.83058530)	31	(0.16926537, 0.83846152)
11	(0.92664230, 0.74431312)	32	(0.15900619, 0.80498970)
12	(0.90189284, 0.69552708)	33	(0.27068156, 0.59086955)
13	(0.92096335, 0.65143716)	34	(0.35037538, 0.51823884)
14	(0.89600557, 0.58111173)	35	(0.51996708, 0.29637322)
15	(0.88191634, 0.57339209)	36	(0.41943192, 0.29763186)
16	(0.80710948, 0.06610739)	37	(0.33229166, 0.40652397)
17	(0.67891032, 0.13654894)	38	(0.35609362, 0.44929621)
18	(0.82150656, 0.24990126)	39	(0.25112855, 0.15176702)
19	(0.85125804, 0.49070349)	40	(0.31141061, 0.15133408)
20	(0.85496521, 0.42990088)	41	(0.33962229, 0.03530093)
21	(0.87710571, 0.39921954)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	25	43.1%
3	25	43.1%
4	8	13.8%

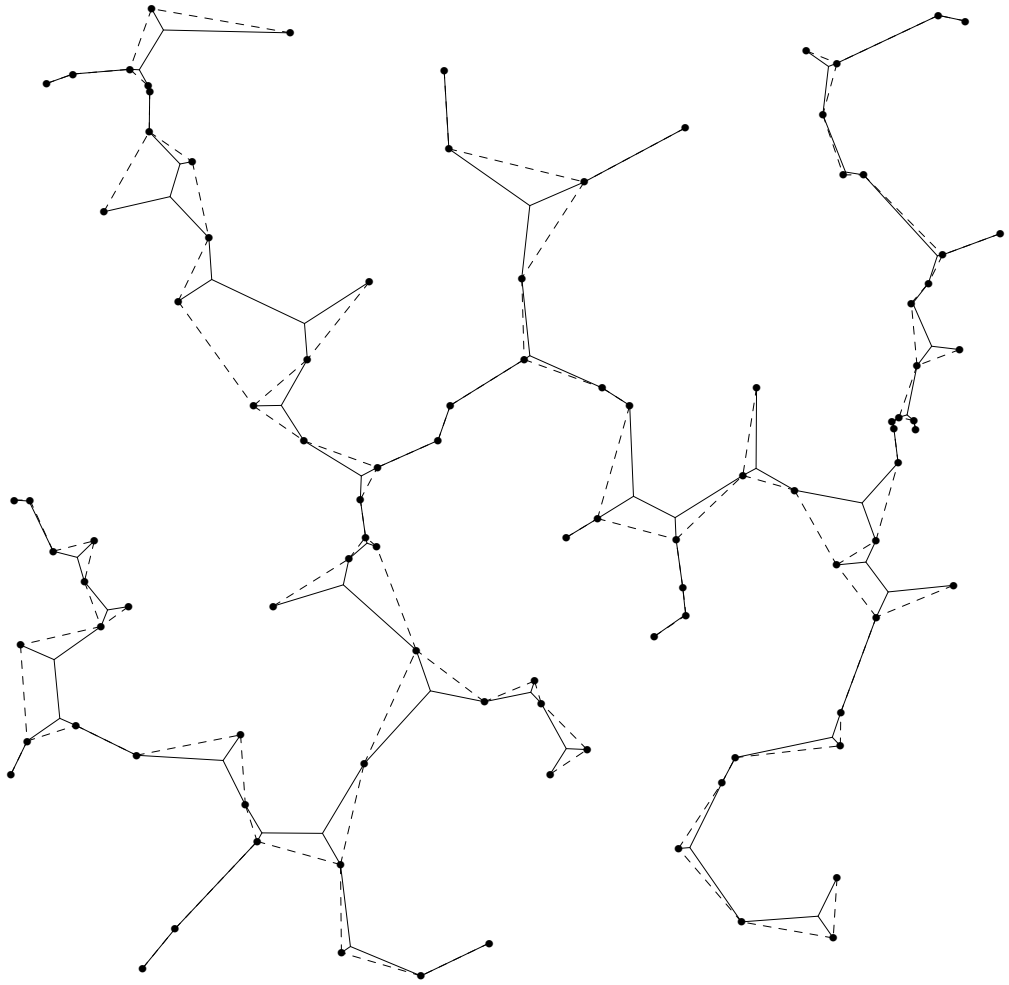


Figure B.1. Cockayne and Hewgill's Test Problem 1 Steiner minimal tree and minimum spanning tree.

B.2 Test Problem 2

Minimum Spanning Tree	6.9352
Steiner Minimal Tree	6.7597
Reduction	2.53%

Given Points

1	(0.16180000, 0.01350000)	51	(0.09230000, 0.52100003)
2	(0.13100000, 0.04660000)	52	(0.14870000, 0.54960001)
3	(0.03960000, 0.06190000)	53	(0.42559999, 0.67830002)
4	(0.01450000, 0.05730000)	54	(0.85799998, 0.21240000)
5	(0.00000000, 0.21760000)	55	(0.06690000, 0.33809999)
6	(0.00240000, 0.36539999)	56	(0.70620000, 0.83910000)
7	(0.06620000, 0.53189999)	57	(0.30500001, 0.78659999)
8	(0.01020000, 0.66250002)	58	(0.36770001, 0.73710001)
9	(0.04240000, 0.71329999)	59	(0.88059998, 0.65600002)
10	(0.02270000, 0.83539999)	60	(0.59990001, 0.38330001)
11	(0.03990000, 0.95109999)	61	(0.51200002, 0.49520001)
12	(0.09440000, 0.95499998)	62	(0.53179997, 0.79350001)
13	(0.24879999, 0.98420000)	63	(0.79369998, 0.50260001)
14	(0.27180001, 0.97680002)	64	(0.72649997, 0.78939998)
15	(0.34369999, 0.91259998)	65	(0.71190000, 0.33750001)
16	(0.42870000, 0.93030000)	66	(0.19320001, 0.74390000)
17	(0.63230002, 0.94690001)	67	(0.47459999, 0.83730000)
18	(0.77130002, 0.99390000)	68	(0.79149997, 0.93419999)
19	(0.94630003, 0.98790002)	69	(0.39179999, 0.60119998)
20	(0.95560002, 0.97189999)	70	(0.56650001, 0.76169997)
21	(0.98140001, 0.85380000)	71	(0.72460002, 0.80260003)
22	(0.99640000, 0.51690000)	72	(0.74610001, 0.16830000)
23	(0.97369999, 0.45649999)	73	(0.90200001, 0.80839998)
24	(0.98860002, 0.36390001)	74	(0.90149999, 0.28240001)
25	(0.98640001, 0.29850000)	75	(0.52679998, 0.71120000)
26	(0.99440002, 0.28839999)	76	(0.38679999, 0.40770000)
27	(0.97850001, 0.17810000)	77	(0.49849999, 0.21370000)
28	(0.85659999, 0.08630000)	78	(0.12549999, 0.50629997)
29	(0.81080002, 0.03680000)	79	(0.81080002, 0.38429999)
30	(0.75709999, 0.03290000)	80	(0.21430001, 0.32960001)
31	(0.68309999, 0.01540000)	81	(0.22149999, 0.39500001)
32	(0.61430001, 0.08740000)	82	(0.39649999, 0.56400001)
33	(0.56730002, 0.09250000)	83	(0.21830000, 0.54530001)
34	(0.54890001, 0.10130000)	84	(0.32130000, 0.63150001)
35	(0.29480001, 0.11490000)	85	(0.33730000, 0.40230000)
36	(0.25290000, 0.18580000)	86	(0.84899998, 0.79979998)
37	(0.57270002, 0.88630003)	87	(0.51730001, 0.46239999)
38	(0.78259999, 0.37630001)	88	(0.17770000, 0.47530001)
39	(0.04980000, 0.20640001)	89	(0.55419999, 0.77020001)
40	(0.75540000, 0.83590001)	90	(0.69239998, 0.70400000)
41	(0.73729998, 0.34020001)	91	(0.38260001, 0.60920000)
42	(0.04650000, 0.20450000)	92	(0.49710000, 0.79479998)
43	(0.52600002, 0.84960002)	93	(0.50449997, 0.59380001)
44	(0.43329999, 0.30160001)	94	(0.31369999, 0.18290000)
45	(0.13150001, 0.39489999)	95	(0.33600000, 0.48289999)
46	(0.89539999, 0.27309999)	96	(0.92909998, 0.19310001)
47	(0.55449998, 0.85619998)	97	(0.75370002, 0.85470003)
48	(0.78619999, 0.54320002)	98	(0.22910000, 0.73879999)
49	(0.52460003, 0.21470000)	99	(0.23880000, 0.64219999)
50	(0.52410001, 0.54240000)	100	(0.76040000, 0.79960001)

Optimal Steiner Points

1	(0.29093367, 0.16742213)	17	(0.62538183, 0.90591252)
2	(0.04927268, 0.94205189)	18	(0.50598818, 0.80247784)
3	(0.97674072, 0.30713668)	19	(0.50109345, 0.82808506)
4	(0.88859707, 0.22475575)	20	(0.42209563, 0.91715449)
5	(0.81813258, 0.15947096)	21	(0.55856514, 0.76198089)
6	(0.84880424, 0.08725557)	22	(0.80139267, 0.38891447)
7	(0.51864254, 0.20858446)	23	(0.14960286, 0.51008642)
8	(0.35268921, 0.41730443)	24	(0.15961896, 0.53608149)
9	(0.30457357, 0.77768844)	25	(0.21663494, 0.39242718)
10	(0.96929866, 0.85895777)	26	(0.17557833, 0.41821027)
11	(0.48313031, 0.66476297)	27	(0.25246188, 0.62370849)
12	(0.37080795, 0.63055527)	28	(0.04665659, 0.20531225)
13	(0.39989758, 0.67879057)	29	(0.02320942, 0.22563149)
14	(0.87203532, 0.78841555)	30	(0.04331886, 0.32993984)
15	(0.72821695, 0.81214023)	31	(0.06065588, 0.07739827)
16	(0.74758643, 0.81529027)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	45	66.2%
3	16	23.5%
4	6	8.8%
5	1	1.5%

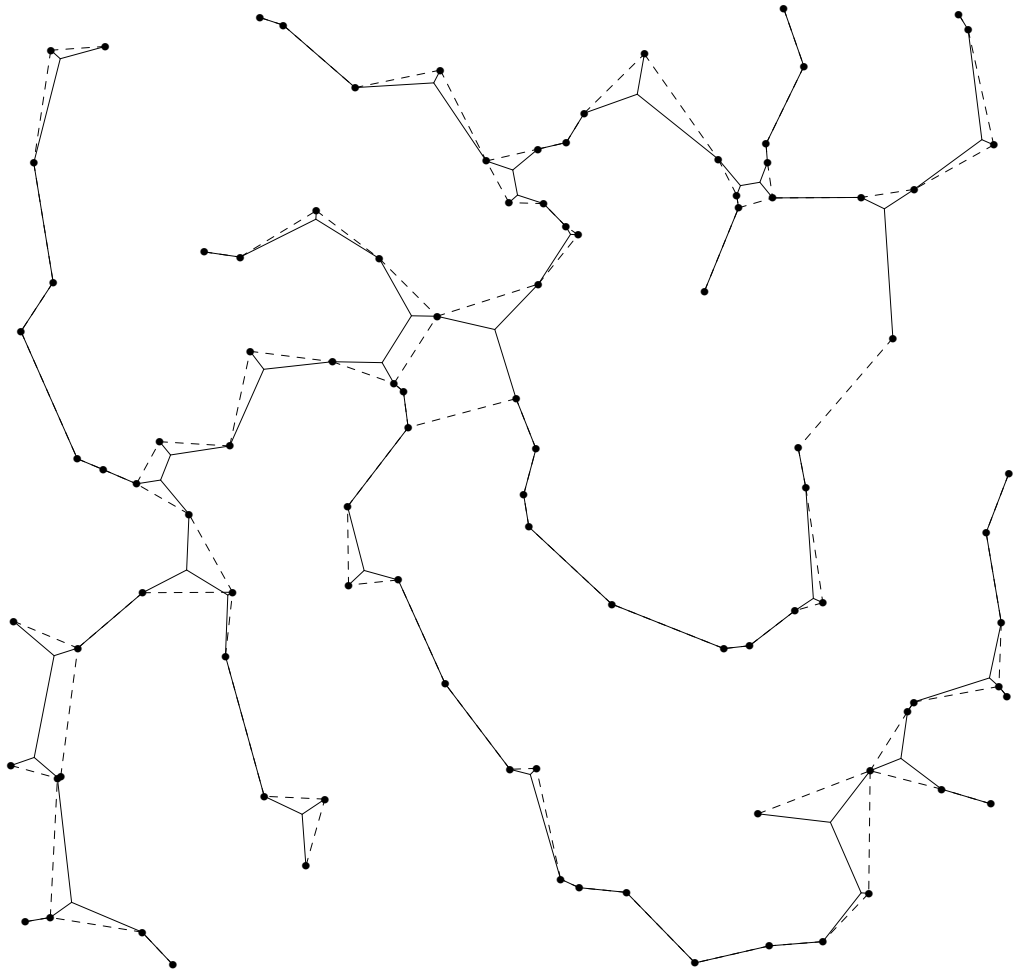


Figure B.2. Cockayne and Hewgill's Test Problem 2 Steiner minimal tree and minimum spanning tree.

B.3 Test Problem 3

Minimum Spanning Tree	6.9238
Steiner Minimal Tree	6.6672
Reduction	3.71%

Given Points

1	(0.45559999, 0.00000000)	51	(0.09980000, 0.73379999)
2	(0.37599999, 0.03050000)	52	(0.79290003, 0.34470001)
3	(0.30390000, 0.01360000)	53	(0.28330001, 0.87570000)
4	(0.20100001, 0.02720000)	54	(0.86519998, 0.14960000)
5	(0.12430000, 0.08460000)	55	(0.74989998, 0.21990000)
6	(0.07530000, 0.15120000)	56	(0.39750001, 0.41700000)
7	(0.00510000, 0.24120000)	57	(0.35450000, 0.80510002)
8	(0.04800000, 0.48500001)	58	(0.93059999, 0.78810000)
9	(0.00300000, 0.76789999)	59	(0.91320002, 0.77969998)
10	(0.01260000, 0.78649998)	60	(0.19820000, 0.42410001)
11	(0.03430000, 0.99680001)	61	(0.10590000, 0.67970002)
12	(0.16540000, 0.98430002)	62	(0.93339998, 0.26850000)
13	(0.19210000, 0.96520001)	63	(0.79809999, 0.79159999)
14	(0.33440000, 0.97350001)	64	(0.10530000, 0.77579999)
15	(0.46110001, 0.91670001)	65	(0.66240001, 0.57830000)
16	(0.46869999, 0.91409999)	66	(0.26740000, 0.73530000)
17	(0.63700002, 0.92390001)	67	(0.50760001, 0.26760000)
18	(0.80350000, 0.93430001)	68	(0.17540000, 0.45420000)
19	(0.81900001, 0.94040000)	69	(0.49550000, 0.43720001)
20	(0.86830002, 0.93820000)	70	(0.36570001, 0.52319998)
21	(0.96390003, 0.86879998)	71	(0.23150000, 0.05170000)
22	(0.95160002, 0.76650000)	72	(0.85530001, 0.38209999)
23	(0.95999998, 0.69840002)	73	(0.61960000, 0.04510000)
24	(0.96770000, 0.66469997)	74	(0.20209999, 0.85509998)
25	(0.99779999, 0.50720000)	75	(0.74390000, 0.65109998)
26	(0.99449998, 0.40660000)	76	(0.58730000, 0.35569999)
27	(0.98119998, 0.29710001)	77	(0.71740001, 0.56809998)
28	(0.99930000, 0.14960000)	78	(0.32319999, 0.18740000)
29	(0.86460000, 0.09530000)	79	(0.60640001, 0.52920002)
30	(0.84920001, 0.05380000)	80	(0.80879998, 0.46869999)
31	(0.59710002, 0.00120000)	81	(0.95810002, 0.33039999)
32	(0.89399999, 0.70359999)	82	(0.22710000, 0.61690003)
33	(0.13730000, 0.43259999)	83	(0.85329998, 0.86290002)
34	(0.29530001, 0.72509998)	84	(0.53839999, 0.57660002)
35	(0.10460000, 0.72899997)	85	(0.35859999, 0.56830001)
36	(0.37259999, 0.66920000)	86	(0.78860003, 0.18730000)
37	(0.53680003, 0.80699998)	87	(0.14030001, 0.76920003)
38	(0.08540000, 0.70080000)	88	(0.28600001, 0.46970001)
39	(0.43630001, 0.62930000)	89	(0.90179998, 0.69900000)
40	(0.91810000, 0.74460000)	90	(0.32859999, 0.47530001)
41	(0.13519999, 0.10160000)	91	(0.80170000, 0.72090000)
42	(0.66960001, 0.26179999)	92	(0.21020000, 0.09600000)
43	(0.15570000, 0.90539998)	93	(0.38600001, 0.22690000)
44	(0.12460000, 0.24510001)	94	(0.63160002, 0.23630001)
45	(0.85839999, 0.81160003)	95	(0.87379998, 0.20870000)
46	(0.75550002, 0.32130000)	96	(0.10150000, 0.42680001)
47	(0.77869999, 0.35920000)	97	(0.12280000, 0.29080001)
48	(0.14450000, 0.40300000)	98	(0.19890000, 0.60280001)
49	(0.13970000, 0.48220000)	99	(0.47620001, 0.30430001)
50	(0.30450001, 0.26850000)	100	(0.69809997, 0.54949999)

Optimal Steiner Points

1	(0.53867543, 0.87091488)	23	(0.19934767, 0.60586357)
2	(0.84923095, 0.92547244)	24	(0.33848494, 0.22184609)
3	(0.90165067, 0.70028985)	25	(0.96597528, 0.30115864)
4	(0.92152095, 0.71504867)	26	(0.92235458, 0.22295967)
5	(0.92523122, 0.77936745)	27	(0.85222131, 0.17978683)
6	(0.93189883, 0.76704460)	28	(0.83238029, 0.38771516)
7	(0.80321515, 0.78849196)	29	(0.79249889, 0.34607786)
8	(0.85223401, 0.81534666)	30	(0.78175431, 0.34871024)
9	(0.60606843, 0.54020363)	31	(0.34826005, 0.47832954)
10	(0.66249675, 0.57508963)	32	(0.45460495, 0.39932650)
11	(0.69790274, 0.55604607)	33	(0.47693610, 0.28386140)
12	(0.09164066, 0.76143008)	34	(0.57728004, 0.29402930)
13	(0.01451531, 0.78001428)	35	(0.63351965, 0.24811485)
14	(0.13676283, 0.77547526)	36	(0.72364020, 0.26304901)
15	(0.18394712, 0.85528731)	37	(0.13276964, 0.42547071)
16	(0.18071969, 0.96383846)	38	(0.15275151, 0.45569706)
17	(0.32214156, 0.87269771)	39	(0.19976702, 0.43507710)
18	(0.35776621, 0.92468733)	40	(0.20639764, 0.09118873)
19	(0.32468343, 0.72948122)	41	(0.22151621, 0.05315994)
20	(0.39487675, 0.62971258)	42	(0.12813175, 0.09961531)
21	(0.09060681, 0.70128471)	43	(0.07120894, 0.20902357)
22	(0.10328748, 0.72887778)	44	(0.59619474, 0.00270064)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	24	43.6%
3	20	36.4%
4	9	16.4%
5	2	3.6%

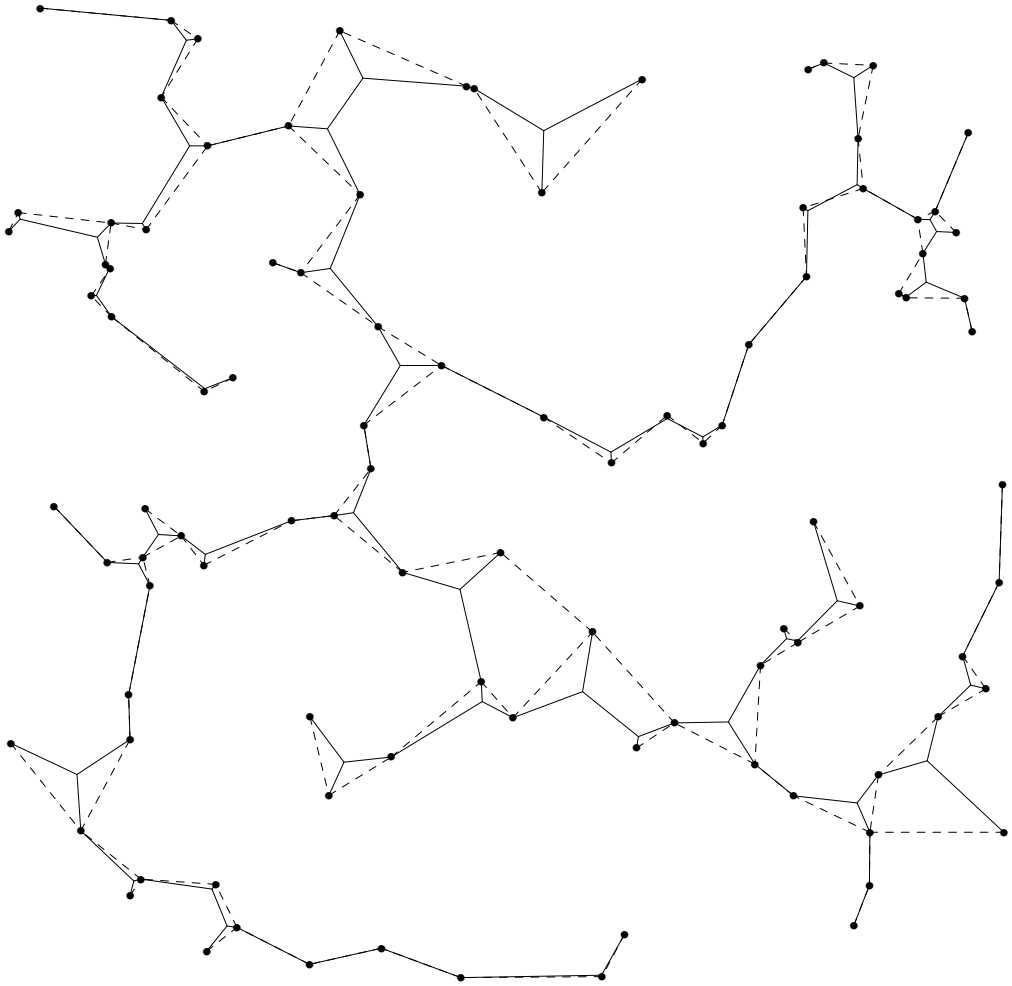


Figure B.3. Cockayne and Hewgill's Test Problem 3 Steiner minimal tree and minimum spanning tree.

B.4 Test Problem 4

Minimum Spanning Tree	6.9214
Steiner Minimal Tree	6.7191
Reduction	2.92%

Given Points

1	(0.69639999, 0.00150000)	51	(0.21290000, 0.12630001)
2	(0.65969998, 0.01550000)	52	(0.32960001, 0.77920002)
3	(0.63020003, 0.01490000)	53	(0.50720000, 0.75650001)
4	(0.44229999, 0.03930000)	54	(0.36600000, 0.73879999)
5	(0.24259999, 0.04580000)	55	(0.34290001, 0.25440001)
6	(0.07960000, 0.04700000)	56	(0.08430000, 0.19310001)
7	(0.06440000, 0.11050000)	57	(0.43439999, 0.29229999)
8	(0.00890000, 0.15400000)	58	(0.51220000, 0.29609999)
9	(0.00000000, 0.21610001)	59	(0.50919998, 0.74839997)
10	(0.01710000, 0.26510000)	60	(0.60509998, 0.85170001)
11	(0.08160000, 0.45379999)	61	(0.35420001, 0.39700001)
12	(0.06060000, 0.48249999)	62	(0.41859999, 0.62699997)
13	(0.04780000, 0.59189999)	63	(0.46050000, 0.46239999)
14	(0.00760000, 0.88470000)	64	(0.54750001, 0.63709998)
15	(0.11030000, 0.97509998)	65	(0.26570001, 0.93820000)
16	(0.23310000, 0.97970003)	66	(0.82840002, 0.59549999)
17	(0.32879999, 0.96799999)	67	(0.35310000, 0.25580001)
18	(0.37689999, 0.97880000)	68	(0.33250001, 0.16630000)
19	(0.46680000, 0.97810000)	69	(0.55419999, 0.19730000)
20	(0.60189998, 0.96910000)	70	(0.82330000, 0.40920001)
21	(0.70620000, 0.99089998)	71	(0.80390000, 0.51150000)
22	(0.79430002, 0.93830001)	72	(0.77679998, 0.06030000)
23	(0.96350002, 0.92250001)	73	(0.38909999, 0.75199997)
24	(0.97450000, 0.93190002)	74	(0.31009999, 0.73729998)
25	(0.98400003, 0.84539998)	75	(0.12120000, 0.45240000)
26	(0.98299998, 0.83870000)	76	(0.07830000, 0.87570000)
27	(0.99900001, 0.71869999)	77	(0.58450001, 0.16660000)
28	(0.98379999, 0.54890001)	78	(0.35380000, 0.73979998)
29	(0.86119998, 0.43939999)	79	(0.15650000, 0.67739999)
30	(0.85259998, 0.42969999)	80	(0.60890001, 0.18860000)
31	(0.83770001, 0.28400001)	81	(0.09470000, 0.86989999)
32	(0.86600000, 0.23019999)	82	(0.48170000, 0.39879999)
33	(0.88550001, 0.21840000)	83	(0.23510000, 0.43340001)
34	(0.93070000, 0.04640000)	84	(0.76239997, 0.07340000)
35	(0.76069999, 0.01060000)	85	(0.29049999, 0.88249999)
36	(0.27149999, 0.66829997)	86	(0.73390001, 0.49399999)
37	(0.71300000, 0.18820000)	87	(0.57709998, 0.48550001)
38	(0.30050001, 0.32069999)	88	(0.45109999, 0.21930000)
39	(0.69749999, 0.79200000)	89	(0.18760000, 0.80180001)
40	(0.76990002, 0.81140000)	90	(0.81279999, 0.88529998)
41	(0.72710001, 0.18850000)	91	(0.33489999, 0.13410001)
42	(0.43770000, 0.21120000)	92	(0.53250003, 0.77929997)
43	(0.20050000, 0.50470001)	93	(0.29330000, 0.26949999)
44	(0.42899999, 0.80809999)	94	(0.73119998, 0.53539997)
45	(0.91939998, 0.72289997)	95	(0.60519999, 0.18050000)
46	(0.18290000, 0.39050001)	96	(0.73769999, 0.21460000)
47	(0.26210001, 0.46730000)	97	(0.73580003, 0.21070001)
48	(0.33890000, 0.22460000)	98	(0.83609998, 0.46460000)
49	(0.38069999, 0.64150000)	99	(0.70319998, 0.16060001)
50	(0.29980001, 0.71499997)	100	(0.63690001, 0.40160000)

Optimal Steiner Points

1	(0.01694267, 0.16282032)	20	(0.74431282, 0.51729685)
2	(0.05007078, 0.15556112)	21	(0.79583317, 0.52263671)
3	(0.08886784, 0.08785835)	22	(0.90017855, 0.62070400)
4	(0.32966626, 0.13785945)	23	(0.96887797, 0.74253535)
5	(0.24810147, 0.10101448)	24	(0.96453840, 0.92220658)
6	(0.06901228, 0.88931602)	25	(0.85625839, 0.43871352)
7	(0.20177956, 0.71434116)	26	(0.58538479, 0.17263238)
8	(0.26811907, 0.68915325)	27	(0.60982716, 0.18613370)
9	(0.33121264, 0.75116420)	28	(0.69915926, 0.17134866)
10	(0.28305689, 0.93141603)	29	(0.72632843, 0.18956751)
11	(0.36323479, 0.73644471)	30	(0.83391106, 0.25470746)
12	(0.50687832, 0.75852168)	31	(0.33879101, 0.25200585)
13	(0.43234602, 0.78707099)	32	(0.30068651, 0.27380386)
14	(0.60698569, 0.83284646)	33	(0.25003257, 0.46293971)
15	(0.41667736, 0.25843641)	34	(0.24541920, 0.43712860)
16	(0.44173592, 0.21891169)	35	(0.31137162, 0.38152492)
17	(0.47925353, 0.31644943)	36	(0.18434252, 0.40911499)
18	(0.48609787, 0.44411170)	37	(0.78335625, 0.04855134)
19	(0.63909978, 0.44543195)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	35	56.5%
3	18	29.0%
4	8	12.9%
5	1	1.6%

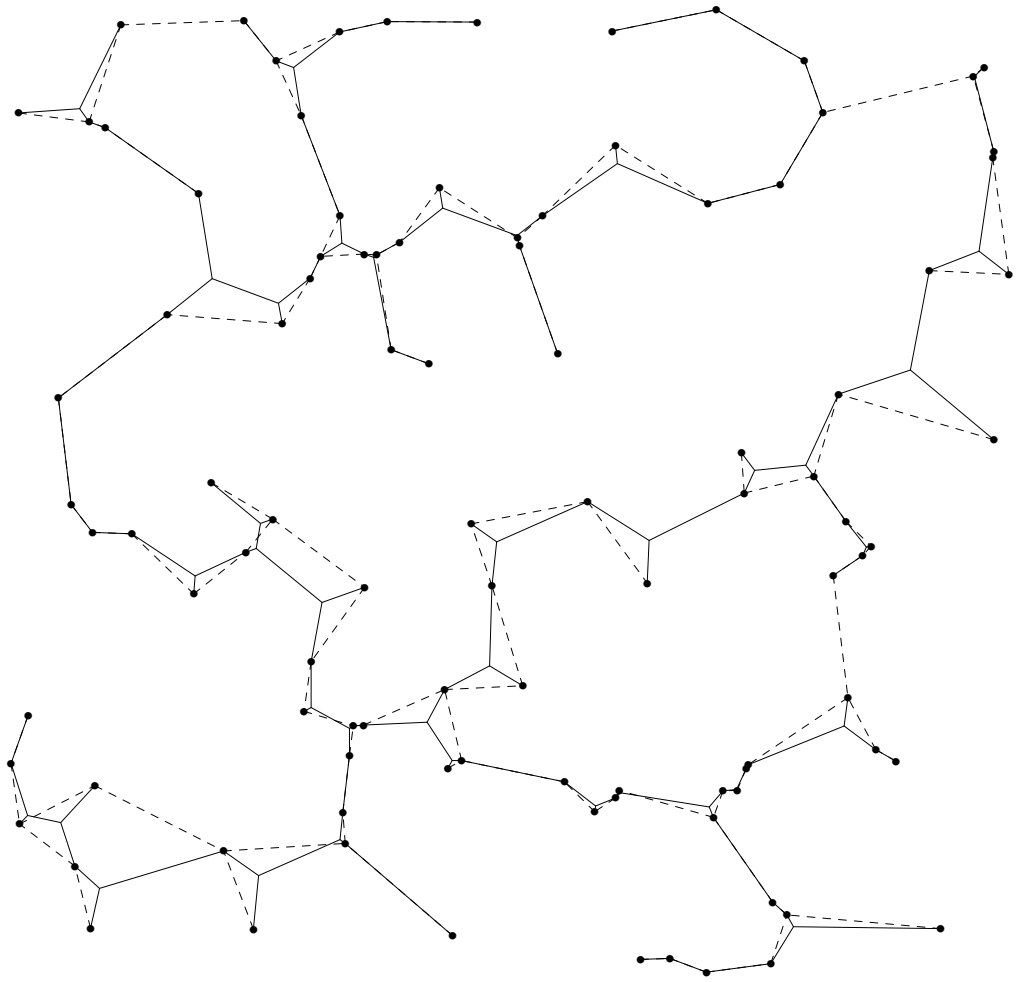


Figure B.4. Cockayne and Hewgill's Test Problem 4 Steiner minimal tree and minimum spanning tree.

B.5 Test Problem 5

Minimum Spanning Tree	6.9352
Steiner Minimal Tree	6.7597
Reduction	2.53%

Given Points

1	(0.16180000, 0.01350000)	51	(0.09230000, 0.52100003)
2	(0.13100000, 0.04660000)	52	(0.14870000, 0.54960001)
3	(0.03960000, 0.06190000)	53	(0.42559999, 0.67830002)
4	(0.01450000, 0.05730000)	54	(0.85799998, 0.21240000)
5	(0.00000000, 0.21760000)	55	(0.06690000, 0.33809999)
6	(0.00240000, 0.36539999)	56	(0.70620000, 0.83910000)
7	(0.06620000, 0.53189999)	57	(0.30500001, 0.78659999)
8	(0.01020000, 0.66250002)	58	(0.36770001, 0.73710001)
9	(0.04240000, 0.71329999)	59	(0.88059998, 0.65600002)
10	(0.02270000, 0.83539999)	60	(0.59990001, 0.38330001)
11	(0.03990000, 0.95109999)	61	(0.51200002, 0.49520001)
12	(0.09440000, 0.95499998)	62	(0.53179997, 0.79350001)
13	(0.24879999, 0.98420000)	63	(0.79369998, 0.50260001)
14	(0.27180001, 0.97680002)	64	(0.72649997, 0.78939998)
15	(0.34369999, 0.91259998)	65	(0.71190000, 0.33750001)
16	(0.42870000, 0.93030000)	66	(0.19320001, 0.74390000)
17	(0.63230002, 0.94690001)	67	(0.47459999, 0.83730000)
18	(0.77130002, 0.99390000)	68	(0.79149997, 0.93419999)
19	(0.94630003, 0.98790002)	69	(0.39179999, 0.60119998)
20	(0.95560002, 0.97189999)	70	(0.56650001, 0.76169997)
21	(0.98140001, 0.85380000)	71	(0.72460002, 0.80260003)
22	(0.99640000, 0.51690000)	72	(0.74610001, 0.16830000)
23	(0.97369999, 0.45649999)	73	(0.90200001, 0.80839998)
24	(0.98860002, 0.36390001)	74	(0.90149999, 0.28240001)
25	(0.98640001, 0.29850000)	75	(0.52679998, 0.71120000)
26	(0.99440002, 0.28839999)	76	(0.38679999, 0.40770000)
27	(0.97850001, 0.17810000)	77	(0.49849999, 0.21370000)
28	(0.85659999, 0.08630000)	78	(0.12549999, 0.50629997)
29	(0.81080002, 0.03680000)	79	(0.81080002, 0.38429999)
30	(0.75709999, 0.03290000)	80	(0.21430001, 0.32960001)
31	(0.68309999, 0.01540000)	81	(0.22149999, 0.39500001)
32	(0.61430001, 0.08740000)	82	(0.39649999, 0.56400001)
33	(0.56730002, 0.09250000)	83	(0.21830000, 0.54530001)
34	(0.54890001, 0.10130000)	84	(0.32130000, 0.63150001)
35	(0.29480001, 0.11490000)	85	(0.33730000, 0.40230000)
36	(0.25290000, 0.18580000)	86	(0.84899998, 0.79979998)
37	(0.57270002, 0.88630003)	87	(0.51730001, 0.46239999)
38	(0.78259999, 0.37630001)	88	(0.17770000, 0.47530001)
39	(0.04980000, 0.20640001)	89	(0.55419999, 0.77020001)
40	(0.75540000, 0.83590001)	90	(0.69239998, 0.70400000)
41	(0.73729998, 0.34020001)	91	(0.38260001, 0.60920000)
42	(0.04650000, 0.20450000)	92	(0.49710000, 0.79479998)
43	(0.52600002, 0.84960002)	93	(0.50449997, 0.59380001)
44	(0.43329999, 0.30160001)	94	(0.31369999, 0.18290000)
45	(0.13150001, 0.39489999)	95	(0.33600000, 0.48289999)
46	(0.89539999, 0.27309999)	96	(0.92909998, 0.19310001)
47	(0.55449998, 0.85619998)	97	(0.75370002, 0.85470003)
48	(0.78619999, 0.54320002)	98	(0.22910000, 0.73879999)
49	(0.52460003, 0.21470000)	99	(0.23880000, 0.64219999)
50	(0.52410001, 0.54240000)	100	(0.76040000, 0.79960001)

Optimal Steiner Points

1	(0.29093367, 0.16742213)	17	(0.62538183, 0.90591252)
2	(0.04927268, 0.94205189)	18	(0.50598818, 0.80247784)
3	(0.97674072, 0.30713668)	19	(0.50109345, 0.82808506)
4	(0.88859707, 0.22475575)	20	(0.42209563, 0.91715449)
5	(0.81813258, 0.15947096)	21	(0.55856514, 0.76198089)
6	(0.84880424, 0.08725557)	22	(0.80139267, 0.38891447)
7	(0.51864254, 0.20858446)	23	(0.14960286, 0.51008642)
8	(0.35268921, 0.41730443)	24	(0.15961896, 0.53608149)
9	(0.30457357, 0.77768844)	25	(0.21663494, 0.39242718)
10	(0.96929866, 0.85895777)	26	(0.17557833, 0.41821027)
11	(0.48313031, 0.66476297)	27	(0.25246188, 0.62370849)
12	(0.37080795, 0.63055527)	28	(0.04665659, 0.20531225)
13	(0.39989758, 0.67879057)	29	(0.02320942, 0.22563149)
14	(0.87203532, 0.78841555)	30	(0.04331886, 0.32993984)
15	(0.72821695, 0.81214023)	31	(0.06065588, 0.07739827)
16	(0.74758643, 0.81529027)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	45	66.2%
3	16	23.5%
4	6	8.8%
5	1	1.5%

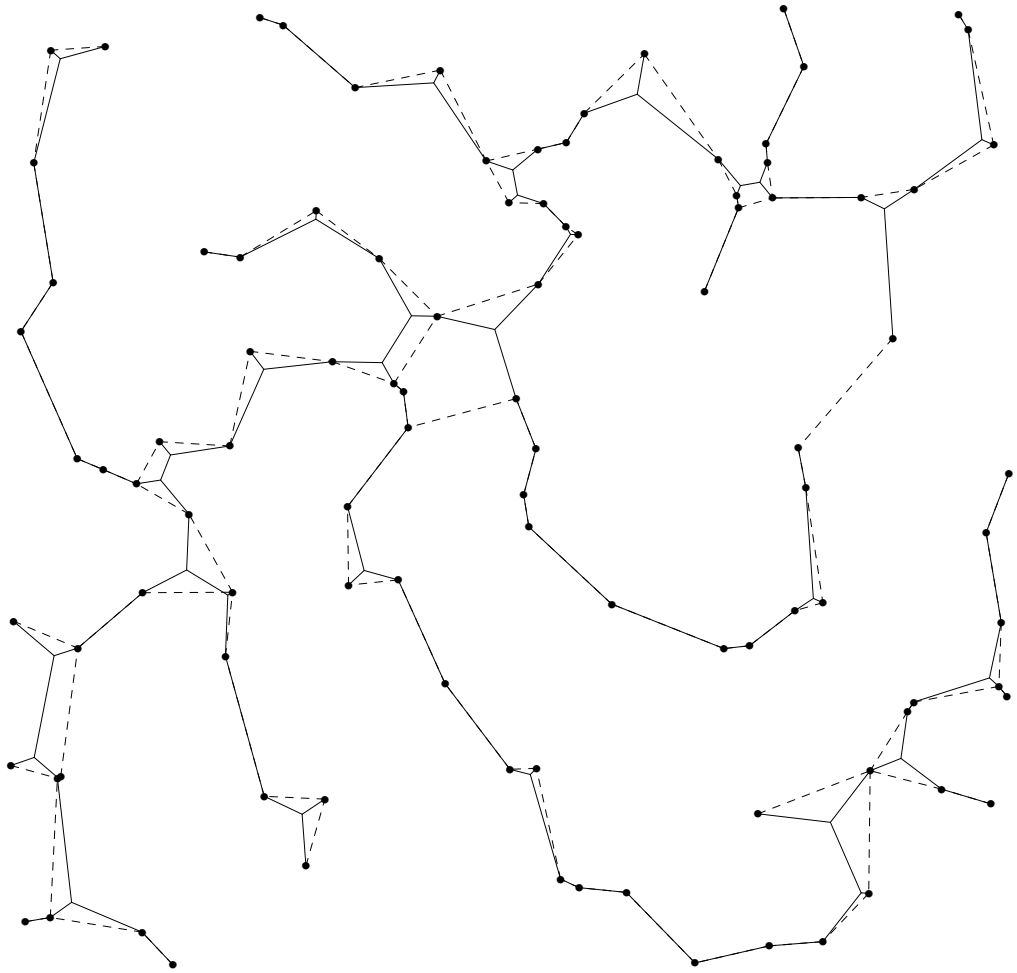


Figure B.5. Cockayne and Hewgill's Test Problem 5 Steiner minimal tree and minimum spanning tree.

B.6 Test Problem 6

Minimum Spanning Tree	6.4843
Steiner Minimal Tree	6.2857
Reduction	3.06%

Given Points

1	(0.59299999, 0.00190000)	51	(0.27820000, 0.80549997)
2	(0.55199999, 0.01930000)	52	(0.50029999, 0.54729998)
3	(0.49090001, 0.04050000)	53	(0.12330000, 0.79540002)
4	(0.47900000, 0.04850000)	54	(0.88480002, 0.82429999)
5	(0.39260000, 0.06470000)	55	(0.70770001, 0.52719998)
6	(0.35810000, 0.02700000)	56	(0.32280001, 0.72079998)
7	(0.17730001, 0.05680000)	57	(0.67760003, 0.61680001)
8	(0.09520000, 0.01350000)	58	(0.76550001, 0.61600000)
9	(0.07000000, 0.05560000)	59	(0.56809998, 0.65730000)
10	(0.08620000, 0.10570000)	60	(0.65679997, 0.74010003)
11	(0.00000000, 0.21870001)	61	(0.62660003, 0.79490000)
12	(0.00120000, 0.29660001)	62	(0.13400000, 0.20590000)
13	(0.07430000, 0.41929999)	63	(0.28600001, 0.16329999)
14	(0.04390000, 0.53950000)	64	(0.06750000, 0.52869999)
15	(0.08510000, 0.73740000)	65	(0.15420000, 0.81750000)
16	(0.07990000, 0.79519999)	66	(0.66920000, 0.75440001)
17	(0.00190000, 0.84230000)	67	(0.36469999, 0.51560003)
18	(0.04440000, 0.90880001)	68	(0.32560000, 0.42850000)
19	(0.12660000, 0.89170003)	69	(0.80580002, 0.53439999)
20	(0.20299999, 0.95130002)	70	(0.33039999, 0.75580001)
21	(0.24670000, 0.94989997)	71	(0.42519999, 0.43340001)
22	(0.52289999, 0.85699999)	72	(0.15290000, 0.13830000)
23	(0.59560001, 0.82770002)	73	(0.56900001, 0.73040003)
24	(0.70260000, 0.85560000)	74	(0.77620000, 0.76480001)
25	(0.85189998, 0.96730000)	75	(0.33309999, 0.40560001)
26	(0.91810000, 0.93800002)	76	(0.15340000, 0.66509998)
27	(0.95520002, 0.76599997)	77	(0.21170001, 0.27520001)
28	(0.99879998, 0.63700002)	78	(0.35969999, 0.76319999)
29	(0.92229998, 0.29780000)	79	(0.81449997, 0.72530001)
30	(0.91630000, 0.23860000)	80	(0.73890001, 0.11530000)
31	(0.98699999, 0.13270000)	81	(0.40840000, 0.17829999)
32	(0.77999997, 0.04090000)	82	(0.49050000, 0.63770002)
33	(0.65240002, 0.05150000)	83	(0.75419998, 0.20540000)
34	(0.48649999, 0.34880000)	84	(0.60589999, 0.56559998)
35	(0.58630002, 0.61049998)	85	(0.40149999, 0.64780003)
36	(0.43460000, 0.50480002)	86	(0.08150000, 0.40799999)
37	(0.51700002, 0.54659998)	87	(0.47819999, 0.28909999)
38	(0.49050000, 0.29800001)	88	(0.35810000, 0.41819999)
39	(0.70590001, 0.64459997)	89	(0.29789999, 0.59609997)
40	(0.81220001, 0.25270000)	90	(0.75520003, 0.47819999)
41	(0.76639998, 0.59719998)	91	(0.24540000, 0.60540003)
42	(0.50880003, 0.60460001)	92	(0.25090000, 0.25520000)
43	(0.78579998, 0.39940000)	93	(0.44549999, 0.27149999)
44	(0.38949999, 0.56620002)	94	(0.35089999, 0.27520001)
45	(0.13609999, 0.15510000)	95	(0.32200000, 0.42530000)
46	(0.79040003, 0.41380000)	96	(0.41920000, 0.69550002)
47	(0.51029998, 0.26030001)	97	(0.44369999, 0.29040000)
48	(0.18490000, 0.27399999)	98	(0.11070000, 0.49480000)
49	(0.18550000, 0.66189998)	99	(0.76020002, 0.47679999)
50	(0.05710000, 0.85979998)	100	(0.72670001, 0.27329999)

Optimal Steiner Points

1	(0.33572266, 0.41354921)	23	(0.33298585, 0.75297123)
2	(0.32473150, 0.42586771)	24	(0.41332847, 0.44219095)
3	(0.75534678, 0.47833419)	25	(0.41974670, 0.49834785)
4	(0.74731833, 0.51467597)	26	(0.38471031, 0.52429247)
5	(0.77835697, 0.54307520)	27	(0.37198681, 0.59632355)
6	(0.76411480, 0.61494327)	28	(0.48730454, 0.29029468)
7	(0.70662832, 0.63894469)	29	(0.09919705, 0.49251819)
8	(0.89941216, 0.93038964)	30	(0.07947394, 0.40817904)
9	(0.81273288, 0.74742651)	31	(0.12623218, 0.19872025)
10	(0.88686061, 0.79849929)	32	(0.01556754, 0.23308896)
11	(0.09310788, 0.78423339)	33	(0.14943054, 0.13783284)
12	(0.04891216, 0.84419721)	34	(0.13738337, 0.10850433)
13	(0.06578566, 0.88224077)	35	(0.08905849, 0.10199737)
14	(0.61003745, 0.59113866)	36	(0.41510034, 0.07981506)
15	(0.51185787, 0.61765742)	37	(0.40570882, 0.24638037)
16	(0.56239378, 0.63293397)	38	(0.28506577, 0.23605503)
17	(0.62677687, 0.76420271)	39	(0.44965112, 0.28355294)
18	(0.65675950, 0.74712169)	40	(0.90294749, 0.25860426)
19	(0.71280932, 0.78942549)	41	(0.76253545, 0.23856151)
20	(0.50824606, 0.55177724)	42	(0.73274797, 0.08064084)
21	(0.41640759, 0.69488448)	43	(0.59275037, 0.00350898)
22	(0.35822287, 0.75853449)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	27	48.2%
3	19	33.9%
4	6	10.7%
5	4	7.1%

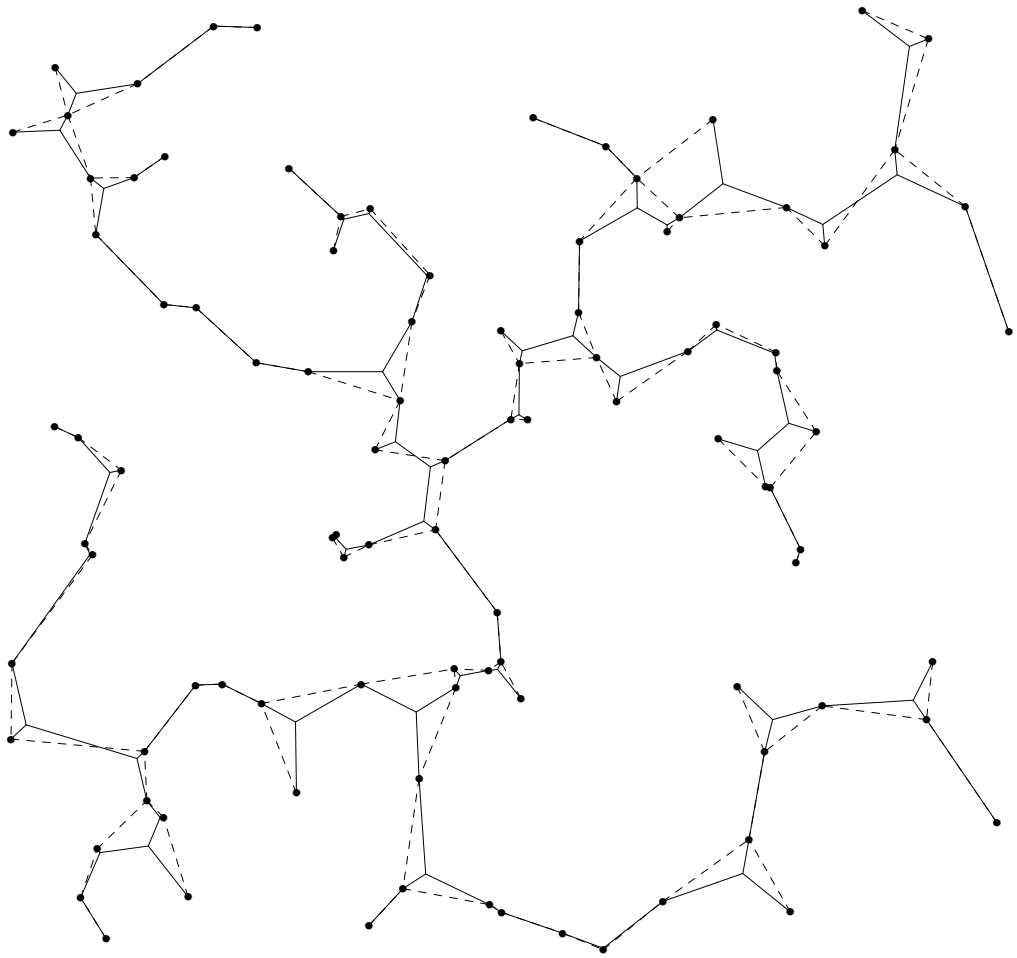


Figure B.6. Cockayne and Hewgill's Test Problem 6 Steiner minimal tree and minimum spanning tree.

B.7 Test Problem 7

Minimum Spanning Tree	6.9062
Steiner Minimal Tree	6.6879
Reduction	3.16%

Given Points

1	(0.00210000, 0.02050000)	51	(0.31720001, 0.44569999)
2	(0.03200000, 0.12980001)	52	(0.16370000, 0.75470001)
3	(0.00000000, 0.22010000)	53	(0.06480000, 0.30919999)
4	(0.02950000, 0.28020000)	54	(0.68430001, 0.61350000)
5	(0.01500000, 0.49610001)	55	(0.54509997, 0.77740002)
6	(0.04280000, 0.75300002)	56	(0.14550000, 0.83350003)
7	(0.00090000, 0.81919998)	57	(0.57330000, 0.43140000)
8	(0.04870000, 0.99210000)	58	(0.82830000, 0.41610000)
9	(0.08760000, 0.99989998)	59	(0.87779999, 0.22409999)
10	(0.12710001, 0.96799999)	60	(0.06260000, 0.29510000)
11	(0.15950000, 0.93669999)	61	(0.56389999, 0.28709999)
12	(0.24690001, 0.95429999)	62	(0.77179998, 0.71749997)
13	(0.44400001, 0.97829998)	63	(0.07050000, 0.25799999)
14	(0.62120003, 0.96590000)	64	(0.15470000, 0.78789997)
15	(0.73430002, 0.95959997)	65	(0.78839999, 0.51490003)
16	(0.86659998, 0.99400002)	66	(0.09590000, 0.51450002)
17	(0.90060002, 0.99910003)	67	(0.40009999, 0.17230000)
18	(0.98170000, 0.87159997)	68	(0.90579998, 0.62180001)
19	(0.98650002, 0.72970003)	69	(0.15220000, 0.79750001)
20	(0.97520000, 0.71039999)	70	(0.40390000, 0.11350000)
21	(0.96399999, 0.63260001)	71	(0.71630001, 0.55350000)
22	(0.94250000, 0.42010000)	72	(0.36539999, 0.49680001)
23	(0.98280001, 0.26449999)	73	(0.20530000, 0.79240000)
24	(0.93900001, 0.17180000)	74	(0.26140001, 0.18750000)
25	(0.89310002, 0.12070000)	75	(0.79460001, 0.46880001)
26	(0.88550001, 0.10050000)	76	(0.43520001, 0.72649997)
27	(0.91000003, 0.02300000)	77	(0.65350002, 0.45760000)
28	(0.78880000, 0.06790000)	78	(0.41980001, 0.50989997)
29	(0.74640000, 0.03670000)	79	(0.21330000, 0.26589999)
30	(0.66670001, 0.02920000)	80	(0.69360000, 0.52380002)
31	(0.61220002, 0.03320000)	81	(0.81440002, 0.60869998)
32	(0.55790001, 0.08040000)	82	(0.08960000, 0.20150000)
33	(0.38589999, 0.06330000)	83	(0.39280000, 0.30329999)
34	(0.14740001, 0.04050000)	84	(0.09930000, 0.10290000)
35	(0.59410000, 0.68959999)	85	(0.68550003, 0.56010002)
36	(0.81930000, 0.79970002)	86	(0.29080001, 0.26740000)
37	(0.51950002, 0.40529999)	87	(0.35670000, 0.57730001)
38	(0.52370000, 0.52569997)	88	(0.33510000, 0.65990001)
39	(0.50080001, 0.50309998)	89	(0.60829997, 0.45780000)
40	(0.15580000, 0.88819999)	90	(0.91149998, 0.17090000)
41	(0.25889999, 0.51429999)	91	(0.56000000, 0.54479998)
42	(0.36269999, 0.66780001)	92	(0.35400000, 0.21400000)
43	(0.52939999, 0.46570000)	93	(0.36289999, 0.63779998)
44	(0.11680000, 0.26859999)	94	(0.38299999, 0.89490002)
45	(0.62000000, 0.49250001)	95	(0.37570000, 0.19780000)
46	(0.65890002, 0.71590000)	96	(0.91109997, 0.81580001)
47	(0.24460000, 0.38640001)	97	(0.32839999, 0.81260002)
48	(0.48379999, 0.46380001)	98	(0.86900002, 0.29409999)
49	(0.34950000, 0.37799999)	99	(0.74260002, 0.58810002)
50	(0.70520002, 0.56669998)	100	(0.13850001, 0.13030000)

Optimal Steiner Points

1	(0.87362486, 0.07515923)	20	(0.16602536, 0.93048739)
2	(0.87952244, 0.22548094)	21	(0.14026660, 0.80811292)
3	(0.91627181, 0.21116635)	22	(0.04784095, 0.77745324)
4	(0.92124808, 0.17871472)	23	(0.16730374, 0.77773273)
5	(0.87432426, 0.38584310)	24	(0.35473636, 0.65736622)
6	(0.60102004, 0.70720589)	25	(0.33905548, 0.81293535)
7	(0.69517171, 0.68715513)	26	(0.39346725, 0.72518665)
8	(0.71077937, 0.58230335)	27	(0.38067153, 0.51587456)
9	(0.77672684, 0.57620311)	28	(0.31908113, 0.47400987)
10	(0.69441074, 0.55463827)	29	(0.31093591, 0.41044563)
11	(0.70503455, 0.56040508)	30	(0.33721021, 0.25944701)
12	(0.64847934, 0.47242680)	31	(0.37603563, 0.30617276)
13	(0.62522483, 0.47707036)	32	(0.25724107, 0.24462408)
14	(0.95839196, 0.63783717)	33	(0.05546237, 0.28094628)
15	(0.95129949, 0.87405115)	34	(0.02995668, 0.27950314)
16	(0.89547378, 0.99167186)	35	(0.08718578, 0.24945484)
17	(0.48382717, 0.47267216)	36	(0.04822616, 0.10236421)
18	(0.50073659, 0.48236597)	37	(0.12925100, 0.13043565)
19	(0.54221827, 0.43529409)	38	(0.11269295, 0.10270358)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	34	55.7%
3	17	27.9%
4	9	14.8%
5	1	1.6%

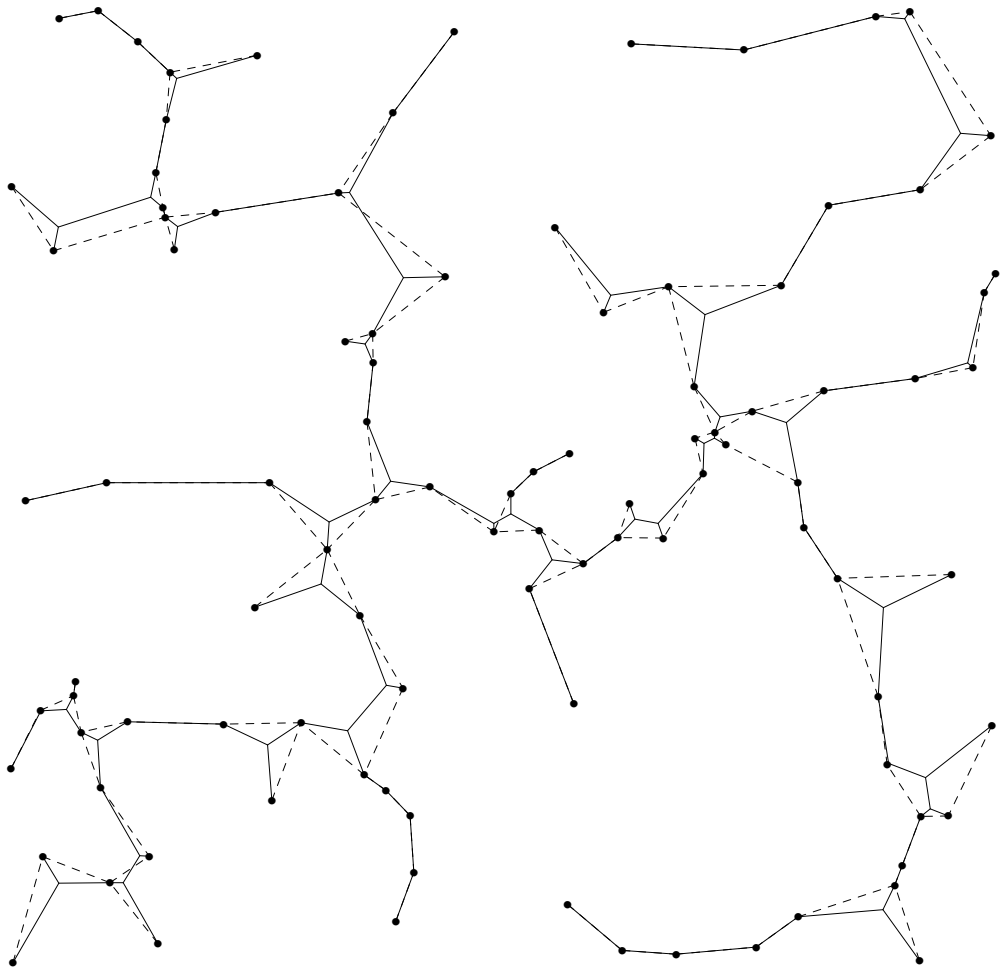


Figure B.7. Cockayne and Hewgill's Test Problem 7 Steiner minimal tree and minimum spanning tree.

B.8 Test Problem 8

Minimum Spanning Tree	6.8273
Steiner Minimal Tree	6.5884
Reduction	3.50%

Given Points

1	(0.43959999, 0.00160000)	51	(0.72180003, 0.05680000)
2	(0.31459999, 0.05160000)	52	(0.29460001, 0.75099999)
3	(0.13349999, 0.14550000)	53	(0.53109998, 0.46280000)
4	(0.10060000, 0.17630000)	54	(0.87639999, 0.61110002)
5	(0.00000000, 0.22139999)	55	(0.28400001, 0.56629997)
6	(0.02180000, 0.48359999)	56	(0.57309997, 0.60790002)
7	(0.04140000, 0.54449999)	57	(0.53310001, 0.08820000)
8	(0.08000000, 0.66360003)	58	(0.84780002, 0.17260000)
9	(0.05930000, 0.73710001)	59	(0.37490001, 0.58630002)
10	(0.10610000, 0.83389997)	60	(0.62279999, 0.83770001)
11	(0.08210000, 0.96890002)	61	(0.83840001, 0.10050000)
12	(0.17919999, 0.99650002)	62	(0.47639999, 0.50749999)
13	(0.22499999, 0.94770002)	63	(0.41819999, 0.87730002)
14	(0.33489999, 0.93690002)	64	(0.25850001, 0.81300002)
15	(0.42580000, 0.95380002)	65	(0.51789999, 0.70050001)
16	(0.48679999, 0.98549998)	66	(0.29820001, 0.39179999)
17	(0.51520002, 0.99870002)	67	(0.85089999, 0.51400000)
18	(0.58859998, 0.96010000)	68	(0.34580001, 0.17280000)
19	(0.60689998, 0.96319997)	69	(0.22360000, 0.56169999)
20	(0.66689998, 0.96420002)	70	(0.86229998, 0.29960001)
21	(0.72079998, 0.80570000)	71	(0.16030000, 0.49550000)
22	(0.79720002, 0.76940000)	72	(0.53009999, 0.26989999)
23	(0.98089999, 0.76220000)	73	(0.22900000, 0.54400003)
24	(0.96920002, 0.73119998)	74	(0.33019999, 0.77450001)
25	(0.95270002, 0.46259999)	75	(0.31900001, 0.71340001)
26	(0.94489998, 0.31110001)	76	(0.72340000, 0.14309999)
27	(0.95940000, 0.08330000)	77	(0.39940000, 0.35490000)
28	(0.92100000, 0.04760000)	78	(0.23220000, 0.19000000)
29	(0.77249998, 0.02840000)	79	(0.15520000, 0.94630003)
30	(0.70160002, 0.02930000)	80	(0.62720001, 0.53560001)
31	(0.57929999, 0.05770000)	81	(0.35190001, 0.73339999)
32	(0.15620001, 0.82139999)	82	(0.60290003, 0.94270003)
33	(0.33649999, 0.30109999)	83	(0.70829999, 0.42580000)
34	(0.21100000, 0.78469998)	84	(0.68140000, 0.41710001)
35	(0.89740002, 0.31680000)	85	(0.72119999, 0.14480001)
36	(0.26609999, 0.72340000)	86	(0.17649999, 0.86049998)
37	(0.58850002, 0.40110001)	87	(0.23999999, 0.51969999)
38	(0.85960001, 0.40950000)	88	(0.86180001, 0.69830000)
39	(0.82980001, 0.27090001)	89	(0.82889998, 0.41710001)
40	(0.77499998, 0.70649999)	90	(0.16689999, 0.31180000)
41	(0.74169999, 0.21740000)	91	(0.67799997, 0.32550001)
42	(0.24180000, 0.60310000)	92	(0.59770000, 0.27350000)
43	(0.76779997, 0.19720000)	93	(0.09650000, 0.36989999)
44	(0.15700001, 0.42170000)	94	(0.33329999, 0.57410002)
45	(0.19040000, 0.87210000)	95	(0.57950002, 0.74010003)
46	(0.15060000, 0.64539999)	96	(0.12670000, 0.57020003)
47	(0.20240000, 0.46239999)	97	(0.78469998, 0.74290001)
48	(0.11290000, 0.82590002)	98	(0.14560001, 0.61650002)
49	(0.55019999, 0.58929998)	99	(0.47889999, 0.61470002)
50	(0.60589999, 0.07880000)	100	(0.45100001, 0.67750001)

Optimal Steiner Points

1	(0.19817641, 0.93626440)	23	(0.81244791, 0.20867825)
2	(0.17409205, 0.95435423)	24	(0.22502176, 0.46797723)
3	(0.11070519, 0.82590252)	25	(0.34768152, 0.34149727)
4	(0.16814695, 0.82499790)	26	(0.31029791, 0.20952050)
5	(0.27730405, 0.74905694)	27	(0.13414976, 0.15018341)
6	(0.24795458, 0.78885150)	28	(0.17991436, 0.46220323)
7	(0.32947022, 0.73277044)	29	(0.12998356, 0.37103060)
8	(0.31785038, 0.75165254)	30	(0.84713322, 0.10542721)
9	(0.14232218, 0.63885260)	31	(0.92114013, 0.06171414)
10	(0.11833809, 0.54983211)	32	(0.72243589, 0.14479275)
11	(0.04492720, 0.53993303)	33	(0.75315571, 0.19728608)
12	(0.40120572, 0.92482287)	34	(0.69892704, 0.08840422)
13	(0.59976608, 0.95661968)	35	(0.57925868, 0.05799438)
14	(0.65877718, 0.33123475)	36	(0.53481758, 0.07596044)
15	(0.64624834, 0.38387197)	37	(0.72337735, 0.04706062)
16	(0.48790532, 0.51003802)	38	(0.88332826, 0.32058579)
17	(0.51019531, 0.58047521)	39	(0.85512090, 0.41719407)
18	(0.46036673, 0.67138112)	40	(0.88097191, 0.46234262)
19	(0.45799515, 0.62835628)	41	(0.78576988, 0.76356685)
20	(0.56934732, 0.59428436)	42	(0.78988409, 0.72035551)
21	(0.22567004, 0.56099081)	43	(0.88044810, 0.68614924)
22	(0.24649400, 0.57914948)	44	(0.63319540, 0.81458122)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	23	41.8%
3	20	36.4%
4	12	21.8%

Figure B.8. Cockayne and Hewgill's Test Problem 8 Steiner minimal tree and minimum spanning tree.

B.9 Test Problem 9

Minimum Spanning Tree	6.5763
Steiner Minimal Tree	6.4001
Reduction	2.68%

Given Points

1	(0.77999997, 0.00390000)	51	(0.71480000, 0.45660001)
2	(0.69430000, 0.00910000)	52	(0.33579999, 0.52999997)
3	(0.43610001, 0.04490000)	53	(0.37459999, 0.71179998)
4	(0.28960001, 0.04250000)	54	(0.80729997, 0.47229999)
5	(0.18089999, 0.04770000)	55	(0.76440001, 0.17470001)
6	(0.08260000, 0.03180000)	56	(0.70039999, 0.84600002)
7	(0.03350000, 0.06360000)	57	(0.30360001, 0.16230001)
8	(0.01140000, 0.09240000)	58	(0.40450001, 0.60399997)
9	(0.00000000, 0.22270000)	59	(0.86580002, 0.73189998)
10	(0.03470000, 0.84930003)	60	(0.79809999, 0.16660000)
11	(0.15899999, 0.92150003)	61	(0.86519998, 0.76859999)
12	(0.24100000, 0.95240003)	62	(0.02790000, 0.09170000)
13	(0.35370001, 0.95429999)	63	(0.48010001, 0.52029997)
14	(0.38290000, 0.95910001)	64	(0.43860000, 0.87870002)
15	(0.65759999, 0.98769999)	65	(0.22800000, 0.56950003)
16	(0.93839997, 0.87529999)	66	(0.88129997, 0.51670003)
17	(0.97049999, 0.83730000)	67	(0.16900000, 0.83209997)
18	(0.89899999, 0.76209998)	68	(0.61430001, 0.79879999)
19	(0.92919999, 0.57480001)	69	(0.29789999, 0.05950000)
20	(0.93239999, 0.49689999)	70	(0.40669999, 0.44100001)
21	(0.96340001, 0.43779999)	71	(0.53490001, 0.79670000)
22	(0.94900000, 0.40400001)	72	(0.37169999, 0.59820002)
23	(0.99949998, 0.30899999)	73	(0.58990002, 0.87059999)
24	(0.99260002, 0.26490000)	74	(0.92150003, 0.25009999)
25	(0.93809998, 0.16429999)	75	(0.78490001, 0.62000000)
26	(0.94300002, 0.12720001)	76	(0.59399998, 0.18960001)
27	(0.96859998, 0.00470000)	77	(0.19390000, 0.14870000)
28	(0.80909997, 0.37009999)	78	(0.75989997, 0.34240001)
29	(0.56790000, 0.12620001)	79	(0.75459999, 0.40070000)
30	(0.24100000, 0.72189999)	80	(0.42750001, 0.16530000)
31	(0.82819998, 0.59140003)	81	(0.46430001, 0.52190000)
32	(0.34320000, 0.28020000)	82	(0.62940001, 0.28940001)
33	(0.76929998, 0.47870001)	83	(0.21830000, 0.56610000)
34	(0.73989999, 0.16960000)	84	(0.48570001, 0.17749999)
35	(0.67299998, 0.27039999)	85	(0.23420000, 0.92360002)
36	(0.20680000, 0.76450002)	86	(0.69000000, 0.32220000)
37	(0.16550000, 0.22849999)	87	(0.90109998, 0.42580000)
38	(0.81050003, 0.38440001)	88	(0.79909998, 0.12080000)
39	(0.91610003, 0.15050000)	89	(0.46470001, 0.51270002)
40	(0.39870000, 0.09100000)	90	(0.51510000, 0.63980001)
41	(0.36149999, 0.21220000)	91	(0.60699999, 0.12540001)
42	(0.32159999, 0.52569997)	92	(0.12130000, 0.29249999)
43	(0.56120002, 0.44940001)	93	(0.63349998, 0.69290000)
44	(0.86710000, 0.54909998)	94	(0.16859999, 0.13060001)
45	(0.07590000, 0.82130003)	95	(0.23940000, 0.56129998)
46	(0.40720001, 0.23000000)	96	(0.44729999, 0.25690001)
47	(0.67089999, 0.47070000)	97	(0.52380002, 0.87000000)
48	(0.57239997, 0.62739998)	98	(0.43200001, 0.49230000)
49	(0.82110000, 0.48320001)	99	(0.41900000, 0.37059999)
50	(0.83920002, 0.23649999)	100	(0.54560000, 0.26830000)

Optimal Steiner Points

1	(0.87475252, 0.75750554)	20	(0.44194028, 0.18372169)
2	(0.95465308, 0.83894163)	21	(0.42430717, 0.22756775)
3	(0.80897844, 0.62119061)	22	(0.56591552, 0.15927322)
4	(0.15724923, 0.84369195)	23	(0.57408518, 0.13287896)
5	(0.17374089, 0.90695834)	24	(0.58425897, 0.25323039)
6	(0.24515720, 0.94628793)	25	(0.66535997, 0.28649887)
7	(0.38203266, 0.95717120)	26	(0.77414429, 0.37225941)
8	(0.56398112, 0.85192251)	27	(0.80589980, 0.37475681)
9	(0.56721228, 0.81998473)	28	(0.74095732, 0.44984269)
10	(0.64318311, 0.89155841)	29	(0.87072378, 0.52110761)
11	(0.46701014, 0.51844132)	30	(0.91069561, 0.52220935)
12	(0.45253432, 0.58678085)	31	(0.94225907, 0.42033026)
13	(0.38805085, 0.61078495)	32	(0.95641232, 0.43872863)
14	(0.01659665, 0.09638540)	33	(0.98890156, 0.26855800)
15	(0.15984429, 0.06384015)	34	(0.84149784, 0.23186667)
16	(0.18769689, 0.15026504)	35	(0.90663511, 0.22774442)
17	(0.27509719, 0.13098447)	36	(0.79332864, 0.16289812)
18	(0.39886186, 0.09102541)	37	(0.92989427, 0.15037653)
19	(0.37191379, 0.23010913)	38	(0.76433778, 0.02218480)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	30	49.2%
3	24	39.3%
4	7	11.5%

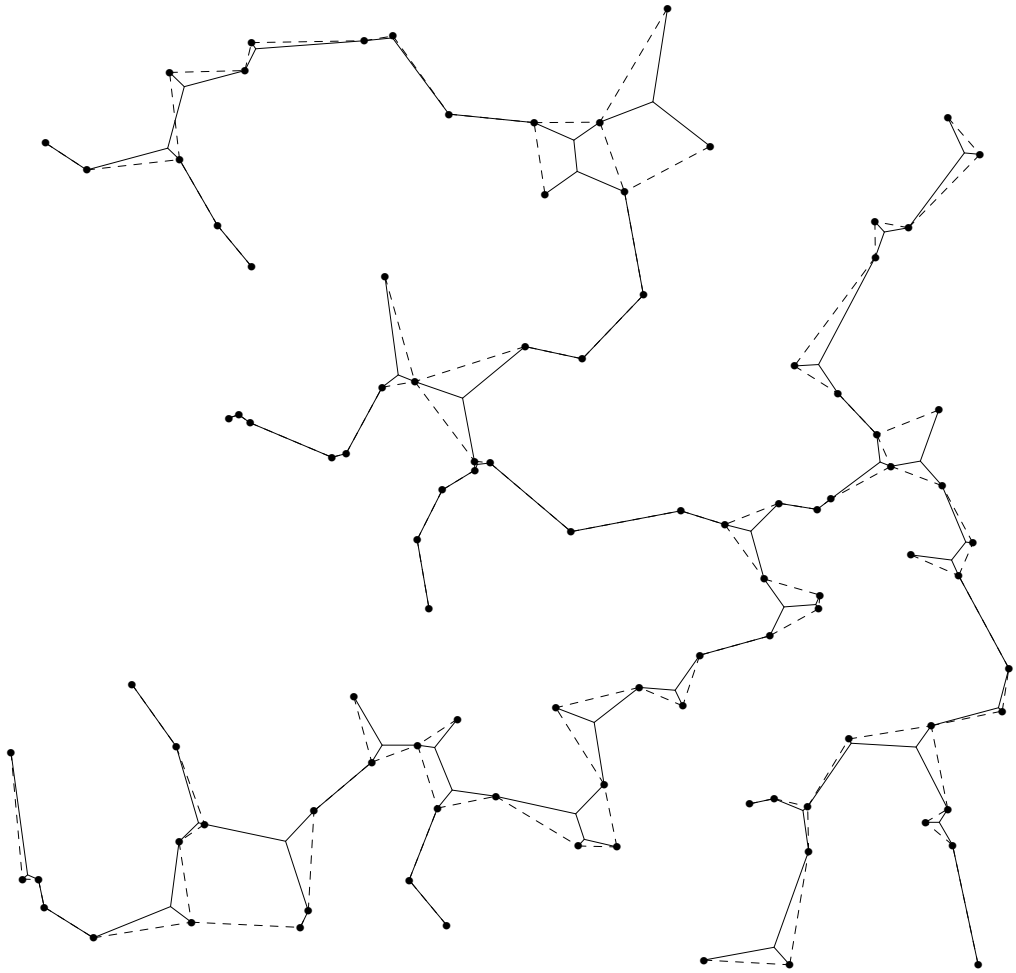


Figure B.9. Cockayne and Hewgill's Test Problem 9 Steiner minimal tree and minimum spanning tree.

B.10 Test Problem 10

Minimum Spanning Tree	6.5572
Steiner Minimal Tree	6.3923
Reduction	2.51%

Given Points

1	(0.63810003, 0.00480000)	51	(0.42100000, 0.64130002)
2	(0.51929998, 0.05460000)	52	(0.85659999, 0.73610002)
3	(0.33000001, 0.00610000)	53	(0.76200002, 0.82620001)
4	(0.24529999, 0.03320000)	54	(0.25119999, 0.08020000)
5	(0.22630000, 0.05220000)	55	(0.46790001, 0.27959999)
6	(0.10060000, 0.05200000)	56	(0.18000001, 0.54200000)
7	(0.02840000, 0.05240000)	57	(0.14320000, 0.19930001)
8	(0.07990000, 0.19130000)	58	(0.34020001, 0.21550000)
9	(0.10120000, 0.23170000)	59	(0.73350000, 0.57359999)
10	(0.09000000, 0.27509999)	60	(0.32699999, 0.65460002)
11	(0.07860000, 0.28060001)	61	(0.28940001, 0.78920001)
12	(0.00730000, 0.36680001)	62	(0.07820000, 0.44250000)
13	(0.03290000, 0.51670003)	63	(0.21860000, 0.21160001)
14	(0.06850000, 0.56449997)	64	(0.44600001, 0.40009999)
15	(0.12340000, 0.65590000)	65	(0.48330000, 0.25760001)
16	(0.06870000, 0.85089999)	66	(0.53320003, 0.29620001)
17	(0.02250000, 0.91149998)	67	(0.26890001, 0.08480000)
18	(0.20400000, 0.90850002)	68	(0.09850000, 0.09510000)
19	(0.27500001, 0.90910000)	69	(0.86159998, 0.61470002)
20	(0.34799999, 0.98070002)	70	(0.72509998, 0.52039999)
21	(0.44790000, 0.97390002)	71	(0.65590000, 0.34819999)
22	(0.51249999, 0.99989998)	72	(0.62019998, 0.38319999)
23	(0.70569998, 0.97539997)	73	(0.65820003, 0.13249999)
24	(0.85540003, 0.91039997)	74	(0.40770000, 0.67850000)
25	(0.92350000, 0.92970002)	75	(0.67100000, 0.72860003)
26	(0.94510001, 0.84179997)	76	(0.17569999, 0.23750000)
27	(0.93000001, 0.80720001)	77	(0.26879999, 0.51929998)
28	(0.91860002, 0.78799999)	78	(0.32820001, 0.54380000)
29	(0.95700002, 0.67159998)	79	(0.62809998, 0.71010000)
30	(0.95550001, 0.61390001)	80	(0.27120000, 0.69319999)
31	(0.95749998, 0.57470000)	81	(0.85740000, 0.77079999)
32	(0.99430001, 0.46830001)	82	(0.61739999, 0.15180001)
33	(0.96840000, 0.38659999)	83	(0.18560000, 0.48769999)
34	(0.88300002, 0.17250000)	84	(0.51539999, 0.85710001)
35	(0.89510000, 0.11990000)	85	(0.44679999, 0.78939998)
36	(0.89920002, 0.09320000)	86	(0.16410001, 0.71200001)
37	(0.85670000, 0.10610000)	87	(0.88150001, 0.41049999)
38	(0.71679997, 0.01710000)	88	(0.44229999, 0.65810001)
39	(0.15830000, 0.23340000)	89	(0.31540000, 0.50110000)
40	(0.69859999, 0.17739999)	90	(0.30620000, 0.82779998)
41	(0.23649999, 0.17690000)	91	(0.35730001, 0.25020000)
42	(0.18250000, 0.24660000)	92	(0.33880001, 0.61269999)
43	(0.11210000, 0.45330000)	93	(0.81419998, 0.78259999)
44	(0.88340002, 0.70920002)	94	(0.37830001, 0.18120000)
45	(0.61189997, 0.36939999)	95	(0.27910000, 0.06540000)
46	(0.87110001, 0.48379999)	96	(0.64200002, 0.80229998)
47	(0.93080002, 0.48230001)	97	(0.52039999, 0.13270000)
48	(0.64150000, 0.36530000)	98	(0.71539998, 0.15210000)
49	(0.73229998, 0.59149998)	99	(0.68500000, 0.70209998)
50	(0.94180000, 0.45429999)	100	(0.74150002, 0.82249999)

Optimal Steiner Points

1	(0.62075537, 0.37538803)	20	(0.19861998, 0.51439369)
2	(0.07207709, 0.86639202)	21	(0.08345477, 0.45234513)
3	(0.27074227, 0.90296102)	22	(0.04316983, 0.51704234)
4	(0.92721069, 0.46887106)	23	(0.03177483, 0.36623207)
5	(0.89477223, 0.46015739)	24	(0.08875221, 0.06517898)
6	(0.96232402, 0.44513059)	25	(0.13906252, 0.21208841)
7	(0.90070587, 0.66072631)	26	(0.10520209, 0.21933006)
8	(0.94143891, 0.65328515)	27	(0.18162698, 0.23995665)
9	(0.91002935, 0.90830535)	28	(0.23462112, 0.05211904)
10	(0.85241640, 0.76694500)	29	(0.88926721, 0.10814950)
11	(0.66459715, 0.79212344)	30	(0.46308586, 0.19806066)
12	(0.72014207, 0.83211845)	31	(0.48986691, 0.30087006)
13	(0.66834891, 0.71969920)	32	(0.47385830, 0.27895787)
14	(0.51697600, 0.84806526)	33	(0.53234792, 0.12234996)
15	(0.34344730, 0.64114875)	34	(0.69966704, 0.15994471)
16	(0.40542886, 0.66459721)	35	(0.75400937, 0.09704275)
17	(0.42252210, 0.65061772)	36	(0.34013134, 0.20352948)
18	(0.25449905, 0.71809423)	37	(0.26456875, 0.16047919)
19	(0.30856073, 0.51579350)	38	(0.26415071, 0.08757840)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	33	54.1%
3	20	32.8%
4	6	9.8%
5	2	3.3%

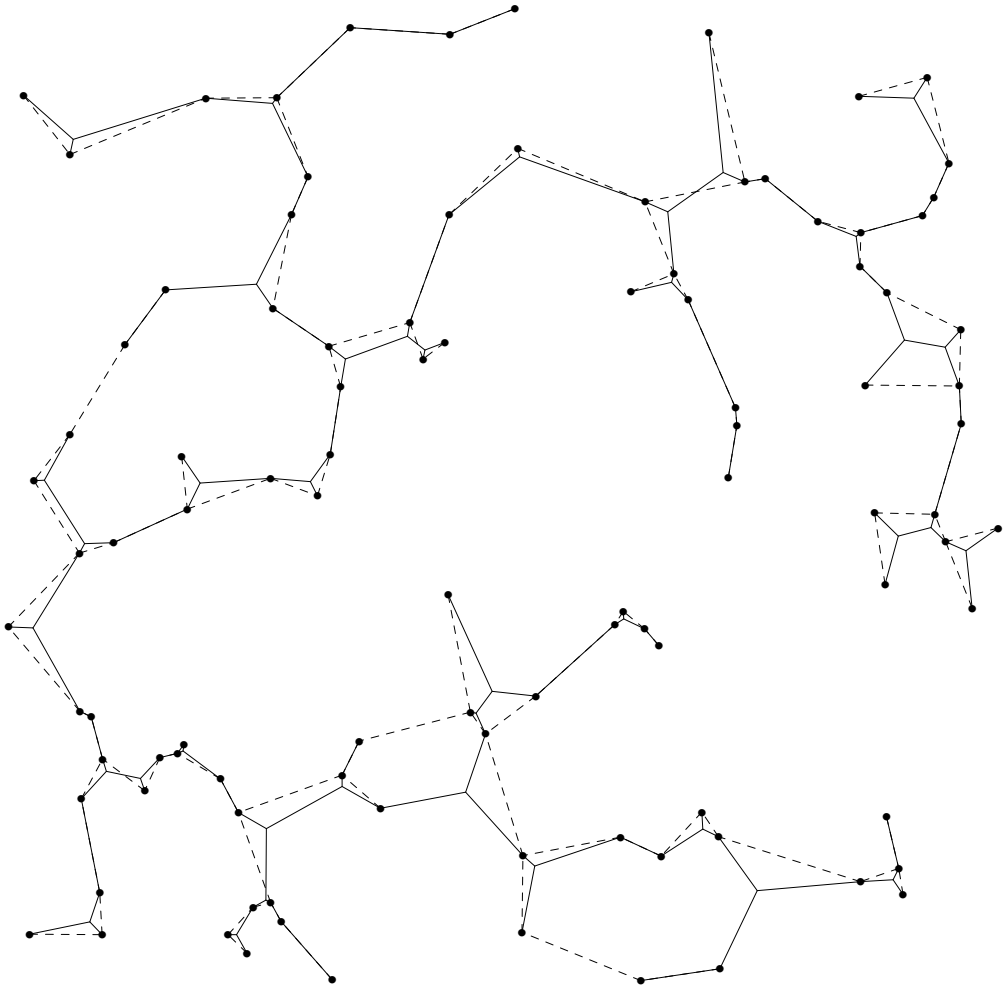


Figure B.10. Cockayne and Hewgill's Test Problem 10 Steiner minimal tree and minimum spanning tree.

B.11 Test Problem 11

Minimum Spanning Tree	6.6482
Steiner Minimal Tree	6.4507
Reduction	2.97%

Given Points

1	(0.02970000, 0.01310000)	51	(0.51590002, 0.12000000)
2	(0.00000000, 0.21770000)	52	(0.22000000, 0.12590000)
3	(0.00010000, 0.33610001)	53	(0.58010000, 0.15700001)
4	(0.05170000, 0.50639999)	54	(0.27640000, 0.85979998)
5	(0.08280000, 0.61680001)	55	(0.21370000, 0.17890000)
6	(0.01490000, 0.87360001)	56	(0.22900000, 0.40869999)
7	(0.08780000, 0.90149999)	57	(0.39969999, 0.12530001)
8	(0.13910000, 0.99330002)	58	(0.23819999, 0.74089998)
9	(0.28410000, 0.93730003)	59	(0.58929998, 0.17560001)
10	(0.37090001, 0.97039998)	60	(0.50389999, 0.83929998)
11	(0.46169999, 0.96010000)	61	(0.33980000, 0.15940000)
12	(0.53350002, 0.94950002)	62	(0.47540000, 0.18040000)
13	(0.70929998, 0.99699998)	63	(0.80330002, 0.56629997)
14	(0.85600001, 0.94849998)	64	(0.54159999, 0.78630000)
15	(0.95850003, 0.95190001)	65	(0.15719999, 0.41589999)
16	(0.98559999, 0.79420000)	66	(0.83179998, 0.69190001)
17	(0.98189998, 0.70179999)	67	(0.84770000, 0.54409999)
18	(0.87510002, 0.61080003)	68	(0.10790000, 0.84039998)
19	(0.88660002, 0.54000002)	69	(0.64829999, 0.45730001)
20	(0.87059999, 0.48600000)	70	(0.56950003, 0.35519999)
21	(0.86280000, 0.45010000)	71	(0.44929999, 0.86949998)
22	(0.87599999, 0.40570000)	72	(0.80379999, 0.35580000)
23	(0.95910001, 0.32949999)	73	(0.84160000, 0.50959998)
24	(0.95940000, 0.32480001)	74	(0.24360000, 0.89700001)
25	(0.88999999, 0.25009999)	75	(0.72119999, 0.83270001)
26	(0.89840001, 0.15760000)	76	(0.73140001, 0.95099998)
27	(0.90009999, 0.02740000)	77	(0.28839999, 0.87140000)
28	(0.83319998, 0.06490000)	78	(0.19340000, 0.78770000)
29	(0.65679997, 0.04480000)	79	(0.10300000, 0.31060001)
30	(0.51260000, 0.02670000)	80	(0.25459999, 0.55089998)
31	(0.50250000, 0.03130000)	81	(0.65329999, 0.47760001)
32	(0.26840001, 0.04960000)	82	(0.60259998, 0.12250000)
33	(0.18189999, 0.03040000)	83	(0.13390000, 0.75349998)
34	(0.46380001, 0.09130000)	84	(0.22460000, 0.21670000)
35	(0.38589999, 0.43439999)	85	(0.57929999, 0.79420000)
36	(0.75349998, 0.28380001)	86	(0.59119999, 0.77499998)
37	(0.46129999, 0.40250000)	87	(0.90219998, 0.81699997)
38	(0.31270000, 0.58289999)	88	(0.85710001, 0.84460002)
39	(0.18260001, 0.46990001)	89	(0.30480000, 0.41440001)
40	(0.41659999, 0.17110001)	90	(0.14260000, 0.48679999)
41	(0.76080000, 0.34000000)	91	(0.25709999, 0.57550001)
42	(0.80100000, 0.92089999)	92	(0.66049999, 0.32890001)
43	(0.73320001, 0.61009997)	93	(0.07450000, 0.28729999)
44	(0.81500000, 0.13970000)	94	(0.81770003, 0.81510001)
45	(0.13920000, 0.88730001)	95	(0.88779998, 0.25299999)
46	(0.24540000, 0.08810000)	96	(0.29760000, 0.43489999)
47	(0.92110002, 0.78619999)	97	(0.11500000, 0.80440003)
48	(0.31510001, 0.71980000)	98	(0.29510000, 0.12370000)
49	(0.30309999, 0.63150001)	99	(0.31150001, 0.64490002)
50	(0.68900001, 0.26719999)	100	(0.33960000, 0.19190000)

Optimal Steiner Points

1	(0.86950064, 0.48561472)	23	(0.11438908, 0.87814867)
2	(0.84278637, 0.54150170)	24	(0.08761436, 0.90042007)
3	(0.86863840, 0.55324525)	25	(0.16406506, 0.91451341)
4	(0.87308180, 0.66990900)	26	(0.30420420, 0.71181905)
5	(0.95078123, 0.71837711)	27	(0.25738716, 0.57528734)
6	(0.94897252, 0.77135211)	28	(0.29872817, 0.59324604)
7	(0.81902921, 0.82767630)	29	(0.08156419, 0.52312082)
8	(0.77763551, 0.85778737)	30	(0.03375977, 0.28492597)
9	(0.78348303, 0.91311318)	31	(0.29470658, 0.42490530)
10	(0.83384788, 0.06527777)	32	(0.18392709, 0.43262628)
11	(0.83357793, 0.12907864)	33	(0.18076441, 0.47667998)
12	(0.89037198, 0.16218986)	34	(0.62258792, 0.36832064)
13	(0.95804143, 0.32633543)	35	(0.69602937, 0.28641126)
14	(0.90899301, 0.31635308)	36	(0.74427372, 0.29483348)
15	(0.86336243, 0.36792225)	37	(0.34577331, 0.16495121)
16	(0.57841259, 0.79213655)	38	(0.39401618, 0.15011202)
17	(0.54227591, 0.78787190)	39	(0.46766376, 0.16846585)
18	(0.46857592, 0.87465572)	40	(0.49177301, 0.12124460)
19	(0.48519829, 0.93658072)	41	(0.47208464, 0.09087266)
20	(0.28207970, 0.87104613)	42	(0.57774627, 0.14303829)
21	(0.26433325, 0.89816087)	43	(0.24469714, 0.06423157)
22	(0.14099927, 0.77605540)	44	(0.24765188, 0.10440020)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	28	50.9%
3	15	27.3%
4	7	12.7%
5	5	9.1%

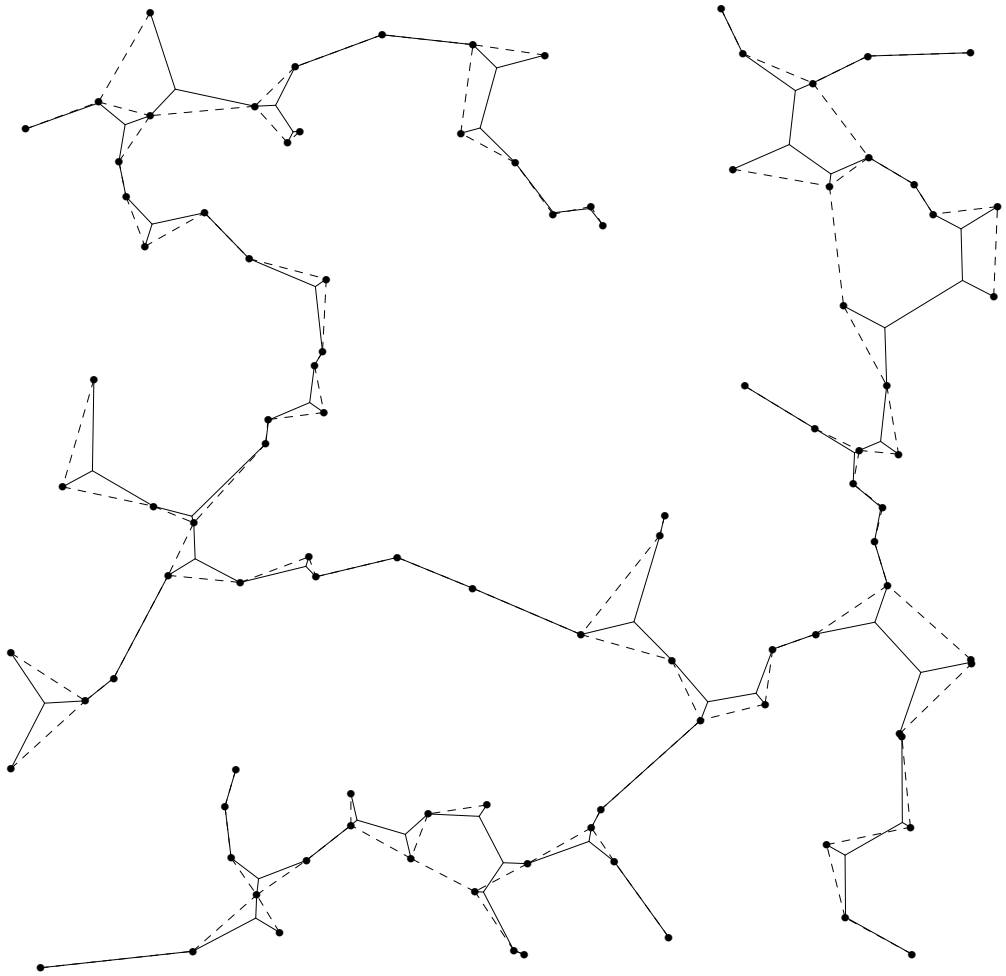


Figure B.11. Cockayne and Hewgill's Test Problem 11 Steiner minimal tree and minimum spanning tree.

B.12 Test Problem 12

Minimum Spanning Tree	6.8170
Steiner Minimal Tree	6.6293
Reduction	2.75%

Given Points

1	(0.92100000, 0.03650000)	51	(0.38929999, 0.79509997)
2	(0.87529999, 0.07180000)	52	(0.54380000, 0.19010000)
3	(0.77429998, 0.06080000)	53	(0.44180000, 0.32960001)
4	(0.72899997, 0.04220000)	54	(0.61350000, 0.38949999)
5	(0.66600001, 0.07610000)	55	(0.48629999, 0.49990001)
6	(0.36939999, 0.06750000)	56	(0.68000001, 0.09120000)
7	(0.24370000, 0.10090000)	57	(0.23000000, 0.28180000)
8	(0.11560000, 0.06580000)	58	(0.76020002, 0.74089998)
9	(0.06510000, 0.07620000)	59	(0.74460000, 0.67040002)
10	(0.03740000, 0.14820001)	60	(0.28830001, 0.68860000)
11	(0.00000000, 0.21210000)	61	(0.56040001, 0.85039997)
12	(0.00530000, 0.36980000)	62	(0.19760001, 0.28160000)
13	(0.05170000, 0.42699999)	63	(0.61559999, 0.13349999)
14	(0.02470000, 0.57050002)	64	(0.46390000, 0.25180000)
15	(0.00770000, 0.69840002)	65	(0.21709999, 0.46529999)
16	(0.07940000, 0.79360002)	66	(0.52539998, 0.71969998)
17	(0.09970000, 0.85710001)	67	(0.41639999, 0.71359998)
18	(0.03030000, 0.97160000)	68	(0.19700000, 0.81750000)
19	(0.13740000, 0.99260002)	69	(0.07120000, 0.34709999)
20	(0.16620000, 0.98140001)	70	(0.18610001, 0.37270001)
21	(0.22840001, 0.98339999)	71	(0.11130000, 0.66710001)
22	(0.28259999, 0.98610002)	72	(0.52960002, 0.88319999)
23	(0.29820001, 0.98970002)	73	(0.47220001, 0.82569999)
24	(0.42070001, 0.95179999)	74	(0.31050000, 0.45609999)
25	(0.59340000, 0.98170000)	75	(0.68010002, 0.64260000)
26	(0.75150001, 0.90679997)	76	(0.06010000, 0.65530002)
27	(0.99449998, 0.99629998)	77	(0.88770002, 0.32030001)
28	(0.88340002, 0.78860003)	78	(0.95310003, 0.19410001)
29	(0.88279998, 0.75900000)	79	(0.43220001, 0.49759999)
30	(0.86989999, 0.69010001)	80	(0.64240003, 0.22300000)
31	(0.87489998, 0.65679997)	81	(0.31320000, 0.12360000)
32	(0.95719999, 0.59520000)	82	(0.49039999, 0.66820002)
33	(0.98720002, 0.58490002)	83	(0.20780000, 0.87050003)
34	(0.95550001, 0.27320001)	84	(0.41389999, 0.23999999)
35	(0.99610001, 0.23019999)	85	(0.84759998, 0.61610001)
36	(0.96149999, 0.18120000)	86	(0.48750001, 0.82179999)
37	(0.40130001, 0.44290000)	87	(0.06230000, 0.62059999)
38	(0.20190001, 0.79909998)	88	(0.28780001, 0.55800003)
39	(0.46320000, 0.80870003)	89	(0.42649999, 0.86900002)
40	(0.58630002, 0.45680001)	90	(0.66240001, 0.34000000)
41	(0.57560003, 0.73329997)	91	(0.31790000, 0.89300001)
42	(0.16840000, 0.88550001)	92	(0.25279999, 0.28290001)
43	(0.19360000, 0.33260000)	93	(0.74659997, 0.61570001)
44	(0.32769999, 0.21170001)	94	(0.43470001, 0.20479999)
45	(0.39980000, 0.42230001)	95	(0.56140000, 0.46460000)
46	(0.16750000, 0.13760000)	96	(0.74180001, 0.82840002)
47	(0.64539999, 0.73930001)	97	(0.18610001, 0.92140001)
48	(0.15930000, 0.58039999)	98	(0.54519999, 0.45030001)
49	(0.13710000, 0.49779999)	99	(0.21089999, 0.24470000)
50	(0.23360001, 0.50639999)	100	(0.85339999, 0.59390002)

Optimal Steiner Points

1	(0.17179719, 0.12091646)	20	(0.03904666, 0.37899083)
2	(0.54588515, 0.45750681)	21	(0.21475445, 0.49330828)
3	(0.87628871, 0.63522017)	22	(0.16461653, 0.51691526)
4	(0.84445387, 0.61118829)	23	(0.07058007, 0.64248502)
5	(0.72997183, 0.64702809)	24	(0.10405570, 0.64795256)
6	(0.68389952, 0.64867067)	25	(0.03138741, 0.69961625)
7	(0.64020312, 0.73099667)	26	(0.26834863, 0.51098847)
8	(0.39985302, 0.91717690)	27	(0.39541882, 0.43658102)
9	(0.33319470, 0.91840178)	28	(0.31265643, 0.12133475)
10	(0.29588544, 0.98585582)	29	(0.32940656, 0.21023753)
11	(0.43236887, 0.82922912)	30	(0.43556979, 0.22629164)
12	(0.46189630, 0.81753498)	31	(0.46345979, 0.24092335)
13	(0.38482198, 0.73511636)	32	(0.60137665, 0.17769372)
14	(0.48702112, 0.68767571)	33	(0.66818655, 0.08667269)
15	(0.53358090, 0.86024660)	34	(0.67986310, 0.09053785)
16	(0.18970349, 0.96511817)	35	(0.72980028, 0.04606974)
17	(0.20571819, 0.86972076)	36	(0.89997959, 0.07205494)
18	(0.18088686, 0.89017415)	37	(0.95471424, 0.19403958)
19	(0.21206191, 0.27242219)	38	(0.97786480, 0.23088256)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	35	57.4%
3	17	27.9%
4	6	9.8%
5	3	4.9%

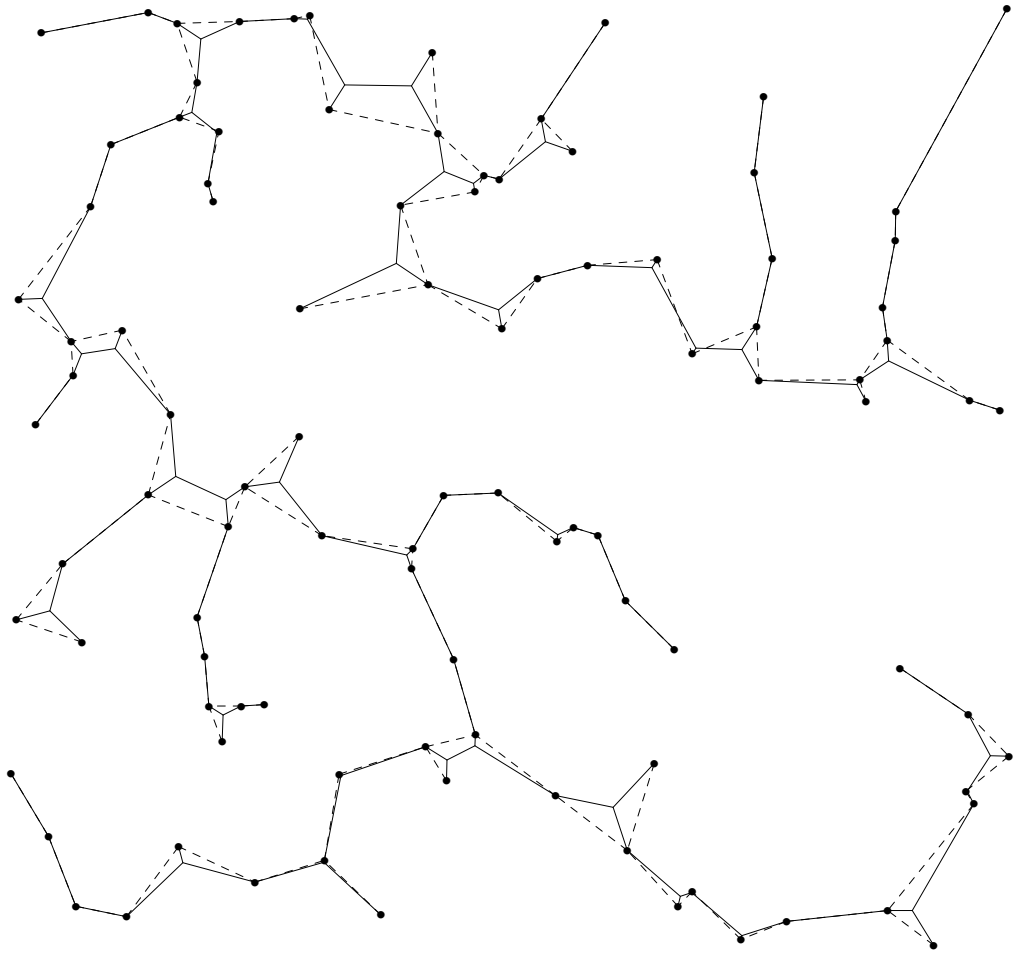


Figure B.12. Cockayne and Hewgill's Test Problem 12 Steiner minimal tree and minimum spanning tree.

B.13 Test Problem 13

Minimum Spanning Tree	6.6141
Steiner Minimal Tree	6.3883
Reduction	3.41%

Given Points

1	(0.59189999, 0.02850000)	51	(0.59359998, 0.20190001)
2	(0.40000001, 0.07500000)	52	(0.72729999, 0.56730002)
3	(0.33300000, 0.09590000)	53	(0.80820000, 0.78250003)
4	(0.27849999, 0.04570000)	54	(0.61799997, 0.39070001)
5	(0.17090000, 0.03010000)	55	(0.13349999, 0.37760001)
6	(0.07900000, 0.12570000)	56	(0.90310001, 0.73290002)
7	(0.06270000, 0.21799999)	57	(0.07840000, 0.42449999)
8	(0.04750000, 0.41800001)	58	(0.16560000, 0.37270001)
9	(0.03050000, 0.43309999)	59	(0.24079999, 0.41270000)
10	(0.02080000, 0.49959999)	60	(0.78850001, 0.80729997)
11	(0.09730000, 0.75449997)	61	(0.55820000, 0.43090001)
12	(0.04220000, 0.93419999)	62	(0.69709998, 0.59969997)
13	(0.08950000, 0.92390001)	63	(0.61510003, 0.61119998)
14	(0.21770000, 0.85030001)	64	(0.27640000, 0.74210000)
15	(0.28520000, 0.85740000)	65	(0.39080000, 0.78140002)
16	(0.53990000, 0.92170000)	66	(0.25209999, 0.11620000)
17	(0.60470003, 0.92129999)	67	(0.35769999, 0.51990002)
18	(0.68769997, 0.90939999)	68	(0.22720000, 0.22290000)
19	(0.85170001, 0.95980000)	69	(0.88770002, 0.33579999)
20	(0.87819999, 0.91750002)	70	(0.30050001, 0.73210001)
21	(0.91979998, 0.73100001)	71	(0.57020003, 0.85039997)
22	(0.97280002, 0.61180001)	72	(0.42060000, 0.23890001)
23	(0.93320000, 0.52800000)	73	(0.59670001, 0.32440001)
24	(0.99339998, 0.40709999)	74	(0.67930001, 0.25510001)
25	(0.94950002, 0.34290001)	75	(0.38640001, 0.62529999)
26	(0.89889997, 0.21990000)	76	(0.80129999, 0.78899997)
27	(0.90039998, 0.04640000)	77	(0.33090001, 0.72680002)
28	(0.84329998, 0.05420000)	78	(0.31920001, 0.44639999)
29	(0.80379999, 0.03920000)	79	(0.53390002, 0.28099999)
30	(0.68220001, 0.03990000)	80	(0.66810000, 0.24720000)
31	(0.63040000, 0.66850001)	81	(0.58990002, 0.08180000)
32	(0.40849999, 0.69730002)	82	(0.27039999, 0.31050000)
33	(0.81300002, 0.40099999)	83	(0.64300001, 0.84689999)
34	(0.68919998, 0.85049999)	84	(0.84930003, 0.26199999)
35	(0.60979998, 0.51270002)	85	(0.68760002, 0.43590000)
36	(0.19980000, 0.79900002)	86	(0.66270000, 0.79390001)
37	(0.39309999, 0.27970001)	87	(0.49349999, 0.43979999)
38	(0.30039999, 0.24789999)	88	(0.65300000, 0.71590000)
39	(0.81089997, 0.34419999)	89	(0.85519999, 0.17690000)
40	(0.70300001, 0.79629999)	90	(0.21709999, 0.84969997)
41	(0.26199999, 0.35910001)	91	(0.59710002, 0.07540000)
42	(0.32949999, 0.23960000)	92	(0.51200002, 0.76969999)
43	(0.55500001, 0.40189999)	93	(0.86650002, 0.51639998)
44	(0.59609997, 0.62210000)	94	(0.34000000, 0.27680001)
45	(0.76480001, 0.14620000)	95	(0.47560000, 0.80570000)
46	(0.88520002, 0.64499998)	96	(0.38139999, 0.59829998)
47	(0.76910001, 0.50760001)	97	(0.75650001, 0.61170000)
48	(0.67790002, 0.41220000)	98	(0.14540000, 0.36710000)
49	(0.12830000, 0.50389999)	99	(0.61960000, 0.81330001)
50	(0.76749998, 0.49689999)	100	(0.22550000, 0.61189997)

Optimal Steiner Points

1	(0.79297870, 0.80690259)	20	(0.67023379, 0.78670573)
2	(0.87106723, 0.91813284)	21	(0.92556107, 0.53500366)
3	(0.28387374, 0.73097122)	22	(0.94033921, 0.60157514)
4	(0.19264634, 0.81387591)	23	(0.89098763, 0.64682299)
5	(0.14051673, 0.81781167)	24	(0.90891272, 0.72757059)
6	(0.32424513, 0.24833225)	25	(0.85209721, 0.31756684)
7	(0.23520909, 0.21986631)	26	(0.87692106, 0.21888469)
8	(0.27929735, 0.25589329)	27	(0.59648716, 0.37816584)
9	(0.17356068, 0.05871882)	28	(0.54992980, 0.42305860)
10	(0.25002393, 0.09386940)	29	(0.63556308, 0.58769339)
11	(0.28094217, 0.07196874)	30	(0.72480363, 0.58789068)
12	(0.09604123, 0.42676166)	31	(0.76962018, 0.49828723)
13	(0.23373502, 0.38896334)	32	(0.80866796, 0.47856015)
14	(0.67855340, 0.85956055)	33	(0.60747707, 0.62277120)
15	(0.63641477, 0.81843281)	34	(0.57932395, 0.28565159)
16	(0.50992191, 0.79025555)	35	(0.61130589, 0.24138406)
17	(0.57177508, 0.83482075)	36	(0.60791427, 0.04948321)
18	(0.57120794, 0.90281272)	37	(0.77027714, 0.14197519)
19	(0.37512287, 0.73550785)	38	(0.76092148, 0.07227451)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	35	57.4%
3	17	27.9%
4	7	11.5%
5	1	1.6%
6	1	1.6%

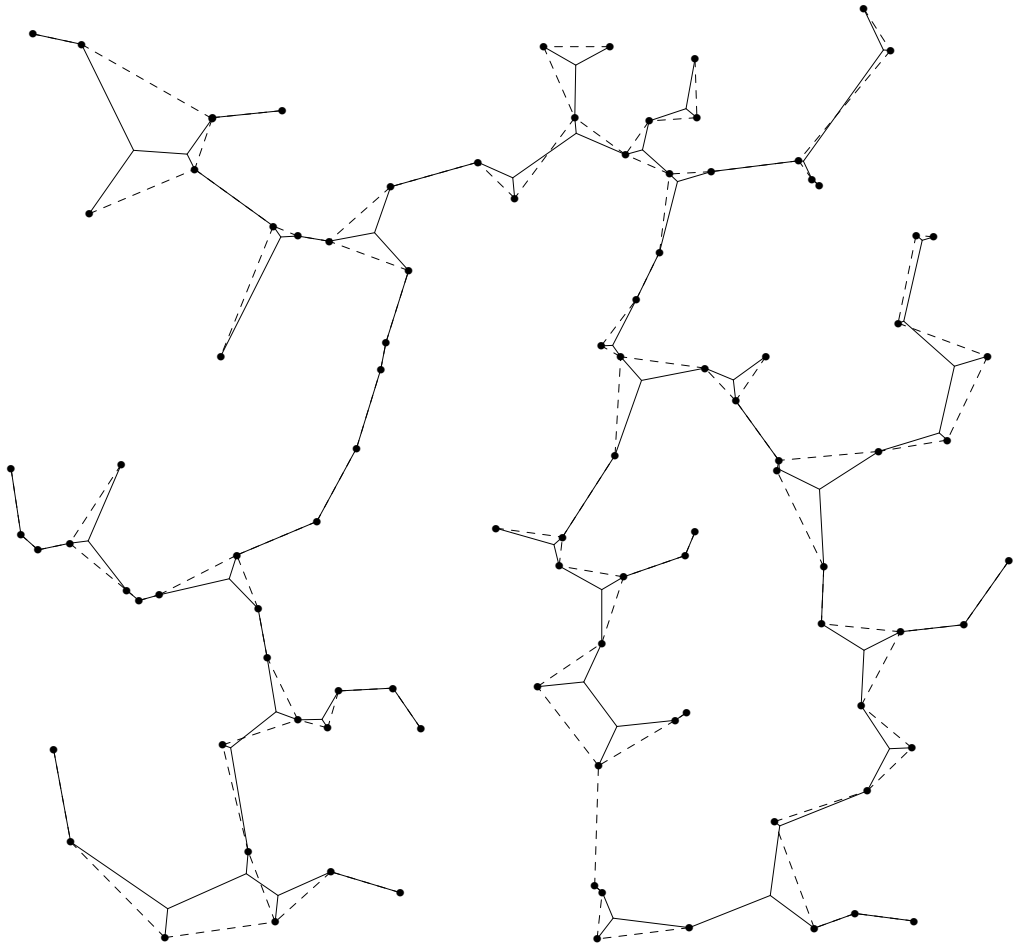


Figure B.13. Cockayne and Hewgill's Test Problem 13 Steiner minimal tree and minimum spanning tree.

B.14 Test Problem 14

Minimum Spanning Tree	6.6418
Steiner Minimal Tree	6.4425
Reduction	3.00%

Given Points

1	(0.92909998, 0.00150000)	51	(0.44880000, 0.64520001)
2	(0.88730001, 0.03430000)	52	(0.43830001, 0.30840001)
3	(0.70440000, 0.04410000)	53	(0.88810003, 0.06960000)
4	(0.62040001, 0.01200000)	54	(0.41790000, 0.72759998)
5	(0.61699998, 0.02010000)	55	(0.35519999, 0.17550001)
6	(0.52219999, 0.03890000)	56	(0.29800001, 0.21780001)
7	(0.46599999, 0.04200000)	57	(0.79049999, 0.76660001)
8	(0.33960000, 0.11910000)	58	(0.20980000, 0.71509999)
9	(0.20060000, 0.06690000)	59	(0.40490001, 0.87080002)
10	(0.10880000, 0.04020000)	60	(0.63990003, 0.53469998)
11	(0.05160000, 0.06460000)	61	(0.91439998, 0.48019999)
12	(0.00130000, 0.13380000)	62	(0.13660000, 0.64099997)
13	(0.01560000, 0.28369999)	63	(0.61589998, 0.27160001)
14	(0.01730000, 0.31779999)	64	(0.52300000, 0.43050000)
15	(0.04780000, 0.50099999)	65	(0.21210000, 0.63370001)
16	(0.05810000, 0.60000002)	66	(0.65530002, 0.36530000)
17	(0.05510000, 0.61960000)	67	(0.34950000, 0.67320001)
18	(0.10520000, 0.75300002)	68	(0.42860001, 0.29780000)
19	(0.04150000, 0.85000002)	69	(0.11710000, 0.90439999)
20	(0.03900000, 0.86540002)	70	(0.53369999, 0.42410001)
21	(0.02010000, 0.96759999)	71	(0.30219999, 0.19650000)
22	(0.06990000, 0.96969998)	72	(0.77139997, 0.45420000)
23	(0.31140000, 0.95840001)	73	(0.15989999, 0.63059998)
24	(0.33080000, 0.96050000)	74	(0.18960001, 0.20510000)
25	(0.36510000, 0.99519998)	75	(0.07880000, 0.12819999)
26	(0.69959998, 0.98920000)	76	(0.93540001, 0.48830000)
27	(0.77300000, 0.98409998)	77	(0.09050000, 0.53939998)
28	(0.91409999, 0.87529999)	78	(0.11670000, 0.09910000)
29	(0.90210003, 0.82849997)	79	(0.21210000, 0.28929999)
30	(0.91060001, 0.80409998)	80	(0.37270001, 0.80729997)
31	(0.98689997, 0.59920001)	81	(0.61589998, 0.37889999)
32	(0.99019998, 0.58420002)	82	(0.60479999, 0.89420003)
33	(0.99000001, 0.39610001)	83	(0.15019999, 0.14860000)
34	(0.89289999, 0.30000001)	84	(0.75919998, 0.72659999)
35	(0.85630000, 0.15660000)	85	(0.37639999, 0.78240001)
36	(0.90130001, 0.07500000)	86	(0.78609997, 0.31840000)
37	(0.37959999, 0.31450000)	87	(0.84160000, 0.33039999)
38	(0.48040000, 0.79949999)	88	(0.51150000, 0.77039999)
39	(0.19620000, 0.21900000)	89	(0.53689998, 0.85259998)
40	(0.70480001, 0.50209999)	90	(0.22120000, 0.32049999)
41	(0.16810000, 0.39309999)	91	(0.64810002, 0.17540000)
42	(0.25790000, 0.21610001)	92	(0.21240000, 0.50610000)
43	(0.39109999, 0.47799999)	93	(0.48030001, 0.23490000)
44	(0.19120000, 0.25920001)	94	(0.35339999, 0.86500001)
45	(0.91280001, 0.41370001)	95	(0.69270003, 0.25979999)
46	(0.19730000, 0.58759999)	96	(0.90130001, 0.50569999)
47	(0.46709999, 0.25459999)	97	(0.53149998, 0.84100002)
48	(0.26370001, 0.69630003)	98	(0.49730000, 0.24390000)
49	(0.71200001, 0.73320001)	99	(0.84759998, 0.69199997)
50	(0.70300001, 0.50239998)	100	(0.43460000, 0.23080000)

Optimal Steiner Points

1	(0.61922485, 0.01925489)	20	(0.29456013, 0.21288851)
2	(0.66707152, 0.05827904)	21	(0.34284887, 0.16939466)
3	(0.14483953, 0.11046796)	22	(0.43102258, 0.29762873)
4	(0.10114908, 0.09746037)	23	(0.45142865, 0.25570801)
5	(0.08825060, 0.06846429)	24	(0.48165902, 0.23982958)
6	(0.02733477, 0.15209441)	25	(0.93260384, 0.58446854)
7	(0.06149560, 0.95988131)	26	(0.98403704, 0.59199613)
8	(0.07853726, 0.91159052)	27	(0.79260927, 0.75665337)
9	(0.08047359, 0.54102814)	28	(0.83694935, 0.74222332)
10	(0.11086824, 0.65395558)	29	(0.90702790, 0.80526251)
11	(0.05703405, 0.61862624)	30	(0.87742752, 0.32913214)
12	(0.19256891, 0.61796516)	31	(0.92084789, 0.39854717)
13	(0.23153998, 0.69006556)	32	(0.91668904, 0.48577607)
14	(0.40686280, 0.68796849)	33	(0.72230566, 0.31595016)
15	(0.41697675, 0.76046836)	34	(0.69038099, 0.36664721)
16	(0.49959019, 0.79741275)	35	(0.73582357, 0.45283377)
17	(0.37332177, 0.85313499)	36	(0.65518659, 0.24308023)
18	(0.32576925, 0.95509577)	37	(0.89262301, 0.07587736)
19	(0.20265977, 0.22651595)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	36	58.1%
3	17	27.4%
4	7	11.3%
5	2	3.2%

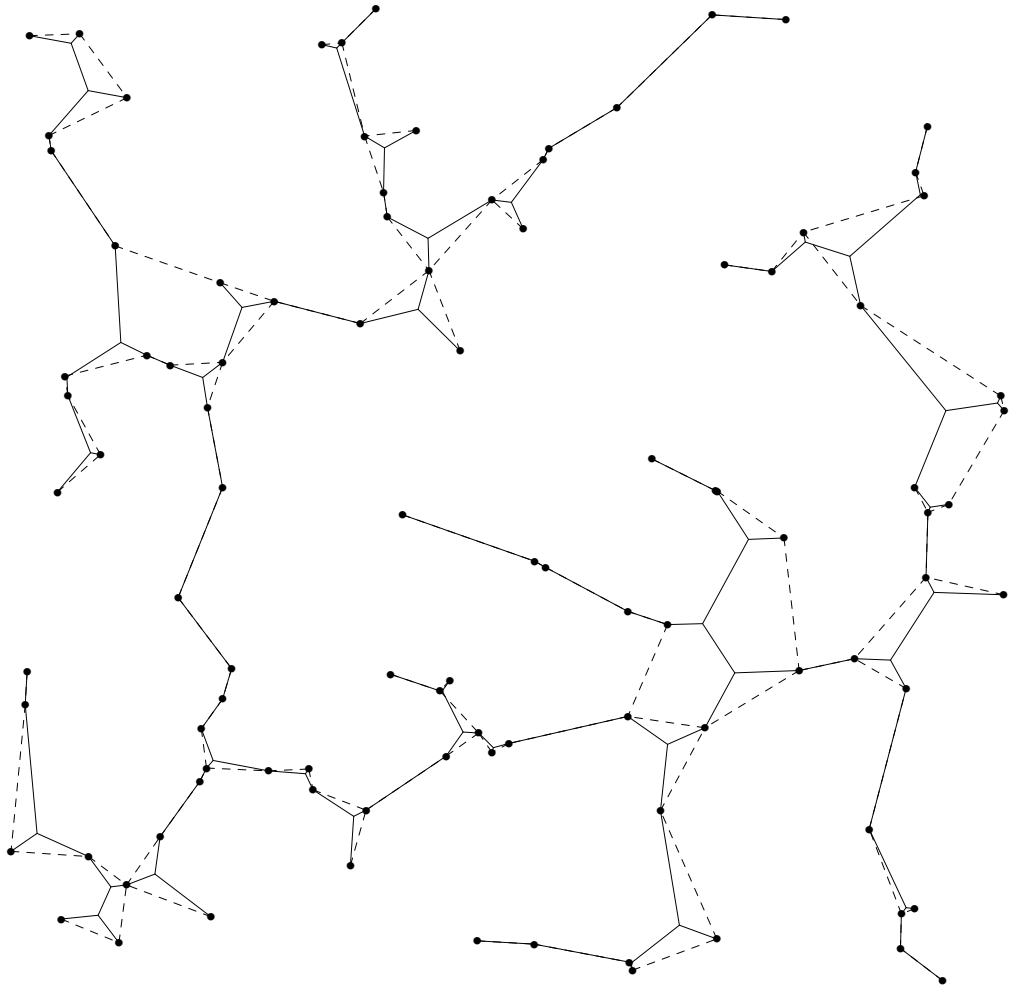


Figure B.14. Cockayne and Hewgill's Test Problem 14 Steiner minimal tree and minimum spanning tree.

B.15 Test Problem 15

Minimum Spanning Tree	6.5720
Steiner Minimal Tree	6.3584
Reduction	3.25%

Given Points

1	(0.28929999, 0.00980000)	51	(0.67439997, 0.11940000)
2	(0.27739999, 0.04670000)	52	(0.41240001, 0.73490000)
3	(0.00000000, 0.21250001)	53	(0.53960001, 0.71399999)
4	(0.02290000, 0.28119999)	54	(0.68239999, 0.93879998)
5	(0.03840000, 0.31680000)	55	(0.77840000, 0.27590001)
6	(0.02460000, 0.47330001)	56	(0.16720000, 0.23140000)
7	(0.00780000, 0.70950001)	57	(0.21310000, 0.38220000)
8	(0.03370000, 0.73030001)	58	(0.29480001, 0.34110001)
9	(0.07190000, 0.75129998)	59	(0.84030002, 0.52710003)
10	(0.10570000, 0.73699999)	60	(0.17540000, 0.45950001)
11	(0.17659999, 0.78640002)	61	(0.45559999, 0.12390000)
12	(0.18449999, 0.81160003)	62	(0.87199998, 0.44729999)
13	(0.21250001, 0.86780000)	63	(0.18770000, 0.38490000)
14	(0.21840000, 0.87510002)	64	(0.76990002, 0.70560002)
15	(0.21370000, 0.89330000)	65	(0.77240002, 0.34349999)
16	(0.24869999, 0.90280002)	66	(0.53520000, 0.67809999)
17	(0.32560000, 0.90450001)	67	(0.33379999, 0.78979999)
18	(0.35530001, 0.90710002)	68	(0.68550003, 0.58829999)
19	(0.46140000, 0.98890001)	69	(0.74290001, 0.90009999)
20	(0.63990003, 0.96670002)	70	(0.26400000, 0.33530000)
21	(0.74489999, 0.97180003)	71	(0.60780001, 0.65050000)
22	(0.80710000, 0.96210003)	72	(0.75489998, 0.18979999)
23	(0.85329998, 0.97000003)	73	(0.31250000, 0.71340001)
24	(0.93229997, 0.87599999)	74	(0.78320003, 0.72140002)
25	(0.93849999, 0.83810002)	75	(0.27550000, 0.49840000)
26	(0.91020000, 0.72560000)	76	(0.71660000, 0.37599999)
27	(0.92060000, 0.71079999)	77	(0.34729999, 0.84899998)
28	(0.96789998, 0.62320000)	78	(0.50080001, 0.62000000)
29	(0.96890002, 0.61989999)	79	(0.56889999, 0.77100003)
30	(0.95270002, 0.57139999)	80	(0.60689998, 0.52749997)
31	(0.92760003, 0.48820001)	81	(0.51520002, 0.79650003)
32	(0.98369998, 0.41470000)	82	(0.60860002, 0.36840001)
33	(0.97270000, 0.40869999)	83	(0.65480000, 0.49380001)
34	(0.96130002, 0.35479999)	84	(0.67629999, 0.38740000)
35	(0.97979999, 0.29429999)	85	(0.70779997, 0.37180001)
36	(0.99129999, 0.10410000)	86	(0.10160000, 0.34779999)
37	(0.98089999, 0.09960000)	87	(0.45600000, 0.70370001)
38	(0.81440002, 0.10580000)	88	(0.77429998, 0.61269999)
39	(0.67369998, 0.07380000)	89	(0.60530001, 0.91310000)
40	(0.64780003, 0.04210000)	90	(0.22499999, 0.54710001)
41	(0.42039999, 0.02820000)	91	(0.19270000, 0.80309999)
42	(0.88880002, 0.78149998)	92	(0.52240002, 0.37760001)
43	(0.31029999, 0.65060002)	93	(0.50880003, 0.14659999)
44	(0.80909997, 0.26060000)	94	(0.59369999, 0.14280000)
45	(0.74150002, 0.49660000)	95	(0.44340000, 0.32229999)
46	(0.59050000, 0.21160001)	96	(0.45469999, 0.85339999)
47	(0.40009999, 0.25299999)	97	(0.52179998, 0.69340003)
48	(0.46570000, 0.44929999)	98	(0.80549997, 0.65249997)
49	(0.47009999, 0.26510000)	99	(0.91619998, 0.31540000)
50	(0.18430001, 0.73060000)	100	(0.52550000, 0.91600001)

Optimal Steiner Points

1	(0.16561541, 0.75071305)	20	(0.34620404, 0.74986994)
2	(0.18836774, 0.80344957)	21	(0.52693605, 0.69327235)
3	(0.21960859, 0.88917702)	22	(0.53985387, 0.66424346)
4	(0.72595912, 0.93545151)	23	(0.55035067, 0.76494300)
5	(0.74734724, 0.96669316)	24	(0.47777942, 0.85453439)
6	(0.64396024, 0.95004195)	25	(0.50893116, 0.91518557)
7	(0.80007327, 0.65276110)	26	(0.06884798, 0.35342219)
8	(0.77294087, 0.70545369)	27	(0.13954440, 0.33736753)
9	(0.89027750, 0.78123158)	28	(0.23295884, 0.51167953)
10	(0.95199317, 0.32384038)	29	(0.17610273, 0.45928067)
11	(0.88243079, 0.47300214)	30	(0.19193812, 0.38880587)
12	(0.92376304, 0.47874552)	31	(0.44415325, 0.27947795)
13	(0.97462314, 0.41343871)	32	(0.49167445, 0.38235345)
14	(0.79053986, 0.54231960)	33	(0.57553017, 0.16218534)
15	(0.66966784, 0.39264622)	34	(0.66758150, 0.11394408)
16	(0.68141735, 0.47274497)	35	(0.75073987, 0.15274358)
17	(0.62915874, 0.52966708)	36	(0.78940022, 0.25935441)
18	(0.65442932, 0.58527493)	37	(0.41564542, 0.03477974)
19	(0.34197149, 0.89731139)	38	(0.29495290, 0.02240902)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	33	54.1%
3	20	32.8%
4	6	9.8%
5	2	3.3%

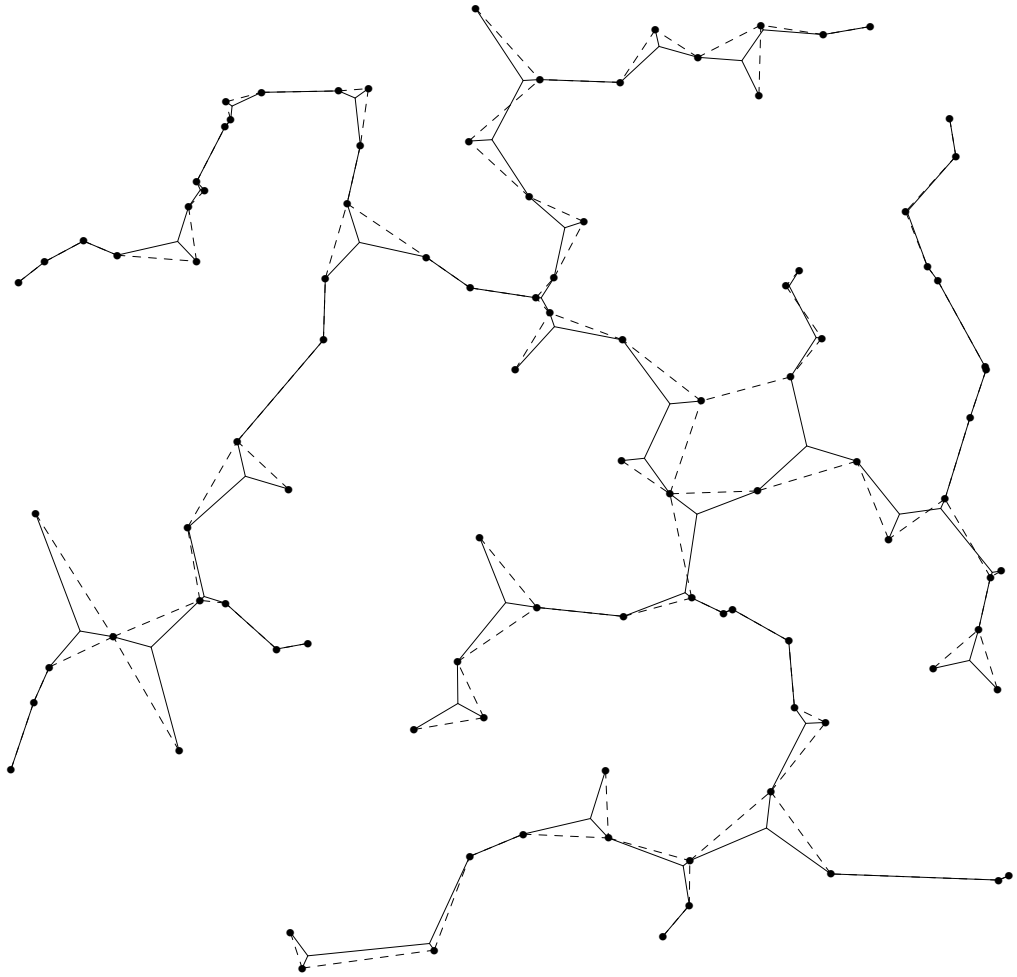


Figure B.15. Cockayne and Hewgill's Test Problem 15 Steiner minimal tree and minimum spanning tree.

B.16 Test Problem 16

Minimum Spanning Tree	6.8599
Steiner Minimal Tree	6.6472
Reduction	3.10%

Given Points

1	(0.55779999, 0.00810000)	51	(0.03470000, 0.49930000)
2	(0.36840001, 0.01550000)	52	(0.46259999, 0.37760001)
3	(0.29499999, 0.01160000)	53	(0.80610001, 0.33870000)
4	(0.25920001, 0.03740000)	54	(0.35089999, 0.29980001)
5	(0.17370000, 0.02770000)	55	(0.21269999, 0.38429999)
6	(0.12260000, 0.01930000)	56	(0.23960000, 0.57069999)
7	(0.07910000, 0.03770000)	57	(0.60189998, 0.11460000)
8	(0.09320000, 0.07220000)	58	(0.23770000, 0.48730001)
9	(0.09430000, 0.07760000)	59	(0.82730001, 0.44409999)
10	(0.00000000, 0.21259999)	60	(0.23130000, 0.81620002)
11	(0.01570000, 0.25549999)	61	(0.88520002, 0.20600000)
12	(0.04230000, 0.26830000)	62	(0.61330003, 0.08220000)
13	(0.04700000, 0.32300001)	63	(0.72570002, 0.27579999)
14	(0.04160000, 0.34529999)	64	(0.43399999, 0.35670000)
15	(0.03940000, 0.45339999)	65	(0.60960001, 0.86140001)
16	(0.02100000, 0.50510001)	66	(0.27820000, 0.63029999)
17	(0.03620000, 0.69410002)	67	(0.90160000, 0.74089998)
18	(0.03130000, 0.75089997)	68	(0.65100002, 0.78789997)
19	(0.03450000, 0.75160003)	69	(0.33829999, 0.70789999)
20	(0.16190000, 0.82880002)	70	(0.19910000, 0.59579998)
21	(0.17560001, 0.99119997)	71	(0.36570001, 0.69029999)
22	(0.41240001, 0.99830002)	72	(0.49720001, 0.74309999)
23	(0.58389997, 0.98960000)	73	(0.72130001, 0.56540000)
24	(0.80729997, 0.99629998)	74	(0.74479997, 0.32260001)
25	(0.85409999, 0.83219999)	75	(0.84380001, 0.54269999)
26	(0.99119997, 0.78839999)	76	(0.46509999, 0.28999999)
27	(0.92710000, 0.57470000)	77	(0.11600000, 0.10320000)
28	(0.95080000, 0.37670001)	78	(0.29339999, 0.40860000)
29	(0.97479999, 0.28749999)	79	(0.80839998, 0.66890001)
30	(0.97909999, 0.24869999)	80	(0.86699998, 0.30660000)
31	(0.93269998, 0.14800000)	81	(0.71160001, 0.10890000)
32	(0.89639997, 0.06020000)	82	(0.80180001, 0.43820000)
33	(0.84490001, 0.04310000)	83	(0.54600000, 0.47520000)
34	(0.62379998, 0.01370000)	84	(0.34070000, 0.77999997)
35	(0.24429999, 0.20680000)	85	(0.64969999, 0.22210000)
36	(0.86839998, 0.48060000)	86	(0.11630000, 0.10400000)
37	(0.79740000, 0.66009998)	87	(0.19870000, 0.09010000)
38	(0.68409997, 0.40210000)	88	(0.10460000, 0.19280000)
39	(0.94290000, 0.28110000)	89	(0.25350001, 0.68820000)
40	(0.46380001, 0.45490000)	90	(0.12520000, 0.37720001)
41	(0.24280000, 0.05190000)	91	(0.64899999, 0.50760001)
42	(0.54420000, 0.18140000)	92	(0.14520000, 0.69910002)
43	(0.58359998, 0.07330000)	93	(0.51959997, 0.28380001)
44	(0.06110000, 0.20160000)	94	(0.08350000, 0.52240002)
45	(0.74559999, 0.19610000)	95	(0.63099998, 0.88330001)
46	(0.35080001, 0.65759999)	96	(0.08640000, 0.29870000)
47	(0.55400002, 0.48170000)	97	(0.59549999, 0.86449999)
48	(0.65149999, 0.50690001)	98	(0.44850001, 0.62970001)
49	(0.66570002, 0.78460002)	99	(0.12930000, 0.34140000)
50	(0.29150000, 0.61530000)	100	(0.36910000, 0.06190000)

Optimal Steiner Points

1	(0.93121392, 0.14778271)	22	(0.16426121, 0.82995051)
2	(0.90277630, 0.27052647)	23	(0.23739937, 0.58237118)
3	(0.97473443, 0.28745160)	24	(0.28028697, 0.61923158)
4	(0.96013677, 0.29382080)	25	(0.43444249, 0.66837698)
5	(0.03926752, 0.49668154)	26	(0.36949486, 0.67985922)
6	(0.35828233, 0.02631549)	27	(0.71050131, 0.23321396)
7	(0.20669086, 0.06125060)	28	(0.74252027, 0.19553700)
8	(0.09019474, 0.04268303)	29	(0.42657542, 0.33702484)
9	(0.09963172, 0.18845604)	30	(0.36087579, 0.32623592)
10	(0.01865311, 0.25388736)	31	(0.25265616, 0.42124522)
11	(0.01935432, 0.22439635)	32	(0.59007788, 0.16710177)
12	(0.06049440, 0.29899660)	33	(0.60443276, 0.08631296)
13	(0.13633560, 0.36557791)	34	(0.58500308, 0.07264921)
14	(0.03371582, 0.75057983)	35	(0.58882886, 0.02992081)
15	(0.04976343, 0.71174568)	36	(0.91086537, 0.77045321)
16	(0.60724813, 0.81135225)	37	(0.78598368, 0.58992016)
17	(0.60574436, 0.85900873)	38	(0.86471474, 0.53252608)
18	(0.62538731, 0.88380754)	39	(0.76725662, 0.34721896)
19	(0.57939118, 0.98320049)	40	(0.75473917, 0.38660765)
20	(0.19872254, 0.72517890)	41	(0.47343564, 0.44713700)
21	(0.20419426, 0.80299270)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	31	53.4%
3	15	25.9%
4	10	17.2%
5	2	3.4%

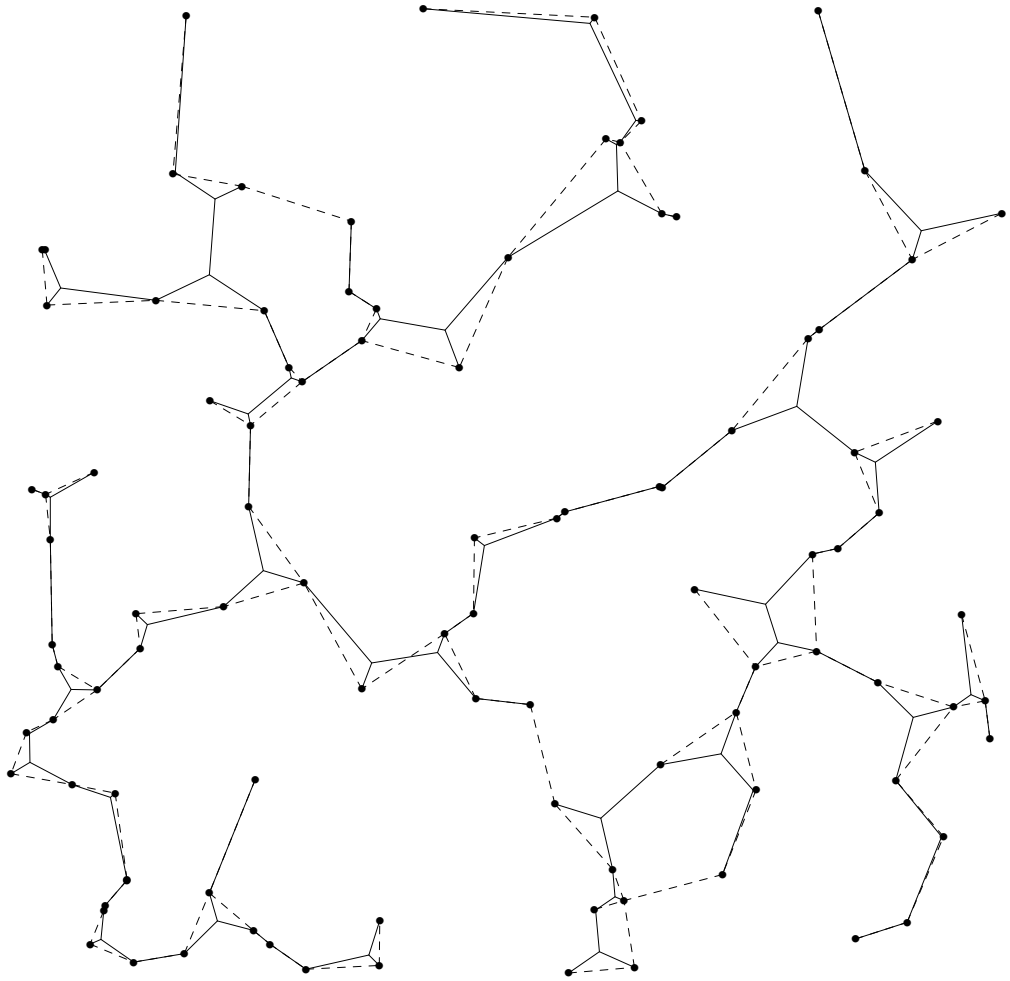


Figure B.16. Cockayne and Hewgill's Test Problem 16 Steiner minimal tree and minimum spanning tree.

B.17 Test Problem 17

Minimum Spanning Tree	6.8022
Steiner Minimal Tree	6.5588
Reduction	3.58%

Given Points

1	(0.02620000, 0.00020000)	51	(0.57179999, 0.72229999)
2	(0.02030000, 0.00710000)	52	(0.40700001, 0.64469999)
3	(0.03920000, 0.09980000)	53	(0.77780002, 0.25130001)
4	(0.00000000, 0.21269999)	54	(0.80519998, 0.48670000)
5	(0.01170000, 0.22220001)	55	(0.37850001, 0.76510000)
6	(0.00960000, 0.36750001)	56	(0.51109999, 0.34900001)
7	(0.00580000, 0.64980000)	57	(0.37259999, 0.81389999)
8	(0.03680000, 0.84859997)	58	(0.20490000, 0.06080000)
9	(0.03400000, 0.93089998)	59	(0.18860000, 0.37779999)
10	(0.04080000, 0.94690001)	60	(0.69630003, 0.41490000)
11	(0.17340000, 0.93959999)	61	(0.36890000, 0.49079999)
12	(0.19720000, 0.93720001)	62	(0.18220000, 0.65719998)
13	(0.34230000, 0.97680002)	63	(0.20379999, 0.40390000)
14	(0.35800001, 0.96289998)	64	(0.86189997, 0.54380000)
15	(0.48469999, 0.99769998)	65	(0.84930003, 0.10300000)
16	(0.54799998, 0.98530000)	66	(0.10010000, 0.79250002)
17	(0.75440001, 0.93360001)	67	(0.55839998, 0.69319999)
18	(0.88239998, 0.95639998)	68	(0.11720000, 0.59329998)
19	(0.90270001, 0.85380000)	69	(0.18210000, 0.09150000)
20	(0.96319997, 0.74959999)	70	(0.50700003, 0.34619999)
21	(0.90109998, 0.67439997)	71	(0.55720001, 0.17410000)
22	(0.92839998, 0.59240001)	72	(0.36849999, 0.87110001)
23	(0.93860000, 0.52969998)	73	(0.79299998, 0.15320000)
24	(0.94559997, 0.46340001)	74	(0.26760000, 0.62110001)
25	(0.95260000, 0.41069999)	75	(0.50029999, 0.84829998)
26	(0.98299998, 0.39660001)	76	(0.75019997, 0.61870003)
27	(0.92820001, 0.15530001)	77	(0.86619997, 0.27059999)
28	(0.90460002, 0.12280000)	78	(0.11150000, 0.69610000)
29	(0.92180002, 0.05070000)	79	(0.29370001, 0.69059998)
30	(0.78140002, 0.05020000)	80	(0.60390002, 0.21940000)
31	(0.74199998, 0.02440000)	81	(0.74510002, 0.66780001)
32	(0.73079997, 0.03880000)	82	(0.80760002, 0.14290000)
33	(0.60060000, 0.08820000)	83	(0.80040002, 0.24670000)
34	(0.49810001, 0.04450000)	84	(0.37599999, 0.50459999)
35	(0.29949999, 0.02030000)	85	(0.12549999, 0.62870002)
36	(0.20029999, 0.05130000)	86	(0.85219997, 0.22490001)
37	(0.07020000, 0.02760000)	87	(0.27660000, 0.32060000)
38	(0.05210000, 0.02260000)	88	(0.67350000, 0.18460000)
39	(0.45510000, 0.11280000)	89	(0.86049998, 0.53899997)
40	(0.54040003, 0.55479997)	90	(0.49320000, 0.78680003)
41	(0.04390000, 0.38029999)	91	(0.50700003, 0.62000000)
42	(0.16750000, 0.85310000)	92	(0.49910000, 0.17690000)
43	(0.29760000, 0.13730000)	93	(0.81180000, 0.67320001)
44	(0.64520001, 0.12890001)	94	(0.39100000, 0.64010000)
45	(0.52329999, 0.51459998)	95	(0.49149999, 0.53250003)
46	(0.63050002, 0.63889998)	96	(0.57599998, 0.29060000)
47	(0.10910000, 0.87629998)	97	(0.45750001, 0.46939999)
48	(0.83260000, 0.10950000)	98	(0.28140000, 0.57319999)
49	(0.90829998, 0.42820001)	99	(0.59979999, 0.34240001)
50	(0.08520000, 0.30540001)	100	(0.38069999, 0.93510002)

Optimal Steiner Points

1	(0.09119984, 0.34134009)	23	(0.93429142, 0.43268394)
2	(0.20038204, 0.38232243)	24	(0.86203146, 0.54365277)
3	(0.00264791, 0.21213831)	25	(0.91606402, 0.55492198)
4	(0.38151488, 0.94710952)	26	(0.89479125, 0.66549194)
5	(0.08493913, 0.85289276)	27	(0.57244426, 0.32221982)
6	(0.05116645, 0.86251813)	28	(0.89260900, 0.10253573)
7	(0.14939879, 0.87945729)	29	(0.60306376, 0.19821894)
8	(0.17617927, 0.93555307)	30	(0.64684570, 0.17058294)
9	(0.13046984, 0.64511192)	31	(0.76162219, 0.19512595)
10	(0.10446084, 0.67285466)	32	(0.78281474, 0.24459948)
11	(0.56774133, 0.65654427)	33	(0.84487087, 0.24169387)
12	(0.52507979, 0.61491698)	34	(0.82301283, 0.11028881)
13	(0.56462979, 0.72011244)	35	(0.74243736, 0.03101423)
14	(0.51829410, 0.52800518)	36	(0.49978837, 0.17563075)
15	(0.45160717, 0.48637578)	37	(0.46114561, 0.11263879)
16	(0.37738493, 0.50061029)	38	(0.31018269, 0.11479595)
17	(0.35029542, 0.69780910)	39	(0.27741355, 0.05980131)
18	(0.39213437, 0.64280647)	40	(0.20286387, 0.06028928)
19	(0.25792682, 0.64576578)	41	(0.17994744, 0.08395286)
20	(0.73332411, 0.64454550)	42	(0.07905475, 0.05864585)
21	(0.86433035, 0.92831200)	43	(0.02585372, 0.00446753)
22	(0.95271540, 0.75036871)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	26	46.4%
3	18	32.1%
4	11	19.6%
5	1	1.8%

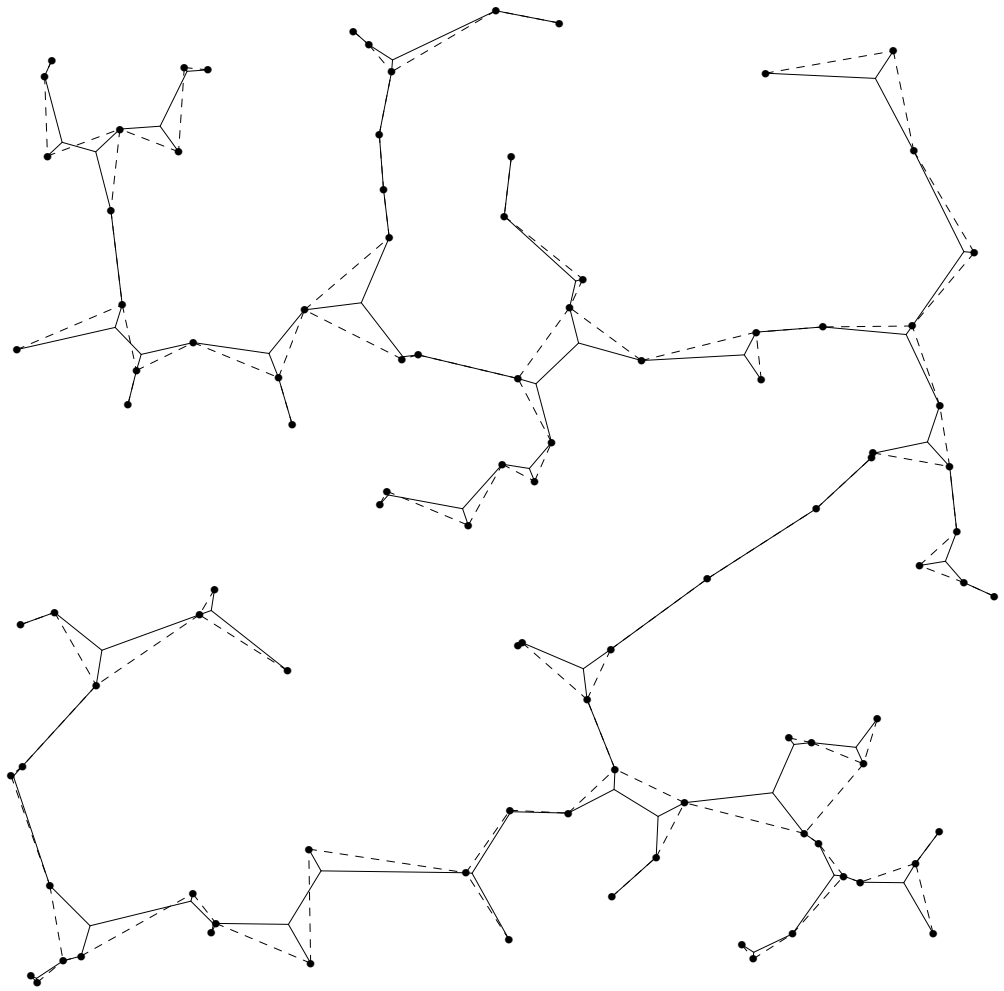


Figure B.17. Cockayne and Hewgill's Test Problem 17 Steiner minimal tree and minimum spanning tree.

B.18 Test Problem 18

Minimum Spanning Tree	6.5206
Steiner Minimal Tree	6.3008
Reduction	3.37%

Given Points

1	(0.53899997, 0.00340000)	51	(0.45800000, 0.17659999)
2	(0.50250000, 0.01420000)	52	(0.47180000, 0.20280001)
3	(0.27849999, 0.04260000)	53	(0.32179999, 0.85939997)
4	(0.17630000, 0.13270000)	54	(0.48890001, 0.52490002)
5	(0.11760000, 0.14010000)	55	(0.67369998, 0.18500000)
6	(0.03610000, 0.21140000)	56	(0.54540002, 0.61019999)
7	(0.00000000, 0.21290000)	57	(0.68019998, 0.81510001)
8	(0.12360000, 0.57440001)	58	(0.34619999, 0.33829999)
9	(0.00900000, 0.70929998)	59	(0.76760000, 0.84869999)
10	(0.05010000, 0.82800001)	60	(0.35040000, 0.72270000)
11	(0.03360000, 0.99010003)	61	(0.28839999, 0.64420003)
12	(0.05280000, 0.98290002)	62	(0.89910001, 0.28049999)
13	(0.10290000, 0.92290002)	63	(0.39430001, 0.69199997)
14	(0.34189999, 0.88940001)	64	(0.11160000, 0.82330000)
15	(0.48580000, 0.88540000)	65	(0.59359998, 0.90840000)
16	(0.52340001, 0.90490001)	66	(0.50099999, 0.81760001)
17	(0.59369999, 0.96109998)	67	(0.52840000, 0.39809999)
18	(0.79860002, 0.94840002)	68	(0.65460002, 0.07370000)
19	(0.95510000, 0.96530002)	69	(0.67250001, 0.23320000)
20	(0.94510001, 0.89920002)	70	(0.63679999, 0.50099999)
21	(0.87239999, 0.76490003)	71	(0.13699999, 0.77980000)
22	(0.86269999, 0.64770001)	72	(0.41069999, 0.32710001)
23	(0.84570003, 0.54140002)	73	(0.64080000, 0.81599998)
24	(0.83329999, 0.45950001)	74	(0.73949999, 0.75760001)
25	(0.88370001, 0.38420001)	75	(0.63209999, 0.16599999)
26	(0.91240001, 0.34299999)	76	(0.24160001, 0.36149999)
27	(0.95480001, 0.29570001)	77	(0.77869999, 0.52289999)
28	(0.98089999, 0.28430000)	78	(0.42280000, 0.62610000)
29	(0.95889997, 0.23520000)	79	(0.43590000, 0.32400000)
30	(0.99750000, 0.15870000)	80	(0.48690000, 0.12780000)
31	(0.93140000, 0.09740000)	81	(0.19430000, 0.54689997)
32	(0.92640001, 0.09500000)	82	(0.42860001, 0.70539999)
33	(0.83759999, 0.06380000)	83	(0.24800000, 0.62940001)
34	(0.71079999, 0.00340000)	84	(0.42359999, 0.26510000)
35	(0.65969998, 0.00370000)	85	(0.54320002, 0.27230000)
36	(0.66570002, 0.58920002)	86	(0.37459999, 0.32580000)
37	(0.51999998, 0.25389999)	87	(0.58209997, 0.80510002)
38	(0.92690003, 0.92919999)	88	(0.15360001, 0.18709999)
39	(0.05430000, 0.70870000)	89	(0.53600001, 0.12830000)
40	(0.46830001, 0.62480003)	90	(0.52770001, 0.31799999)
41	(0.30199999, 0.58029997)	91	(0.69910002, 0.59670001)
42	(0.46720001, 0.58039999)	92	(0.84729999, 0.34450001)
43	(0.83539999, 0.27210000)	93	(0.49649999, 0.42640001)
44	(0.19499999, 0.52609998)	94	(0.33680001, 0.76080000)
45	(0.91270000, 0.21150000)	95	(0.21900000, 0.26690000)
46	(0.72630000, 0.13609999)	96	(0.59820002, 0.48050001)
47	(0.51550001, 0.25220001)	97	(0.64289999, 0.03460000)
48	(0.26820001, 0.27309999)	98	(0.39980000, 0.79850000)
49	(0.35910001, 0.12500000)	99	(0.24280000, 0.31959999)
50	(0.21799999, 0.37230000)	100	(0.56779999, 0.74949998)

Optimal Steiner Points

1	(0.58223003, 0.92000908)	20	(0.50690854, 0.42752105)
2	(0.93058652, 0.92894292)	21	(0.52693975, 0.47282496)
3	(0.81125808, 0.92728889)	22	(0.48032719, 0.60647428)
4	(0.88254189, 0.36242262)	23	(0.41252196, 0.68792278)
5	(0.92281324, 0.30424669)	24	(0.42957523, 0.63346475)
6	(0.97006816, 0.28112596)	25	(0.35440007, 0.70460629)
7	(0.97034776, 0.15966156)	26	(0.53952342, 0.27329788)
8	(0.94339454, 0.21041310)	27	(0.11826074, 0.14413092)
9	(0.08890212, 0.84224159)	28	(0.14842412, 0.15552336)
10	(0.12866855, 0.78097630)	29	(0.25427991, 0.33909333)
11	(0.07035283, 0.70643353)	30	(0.24515334, 0.28442648)
12	(0.18920015, 0.56168514)	31	(0.47369733, 0.23160222)
13	(0.27777606, 0.62784004)	32	(0.42513898, 0.31819615)
14	(0.35895962, 0.79871637)	33	(0.45400131, 0.15685177)
15	(0.32435900, 0.85938638)	34	(0.65994310, 0.00410799)
16	(0.47984126, 0.83326852)	35	(0.66464078, 0.16749728)
17	(0.56296986, 0.79086119)	36	(0.68593615, 0.13424276)
18	(0.72558826, 0.80428237)	37	(0.51112837, 0.11387512)
19	(0.81811023, 0.51297057)	38	(0.51131415, 0.01931197)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	31	50.8%
3	22	36.1%
4	8	13.1%

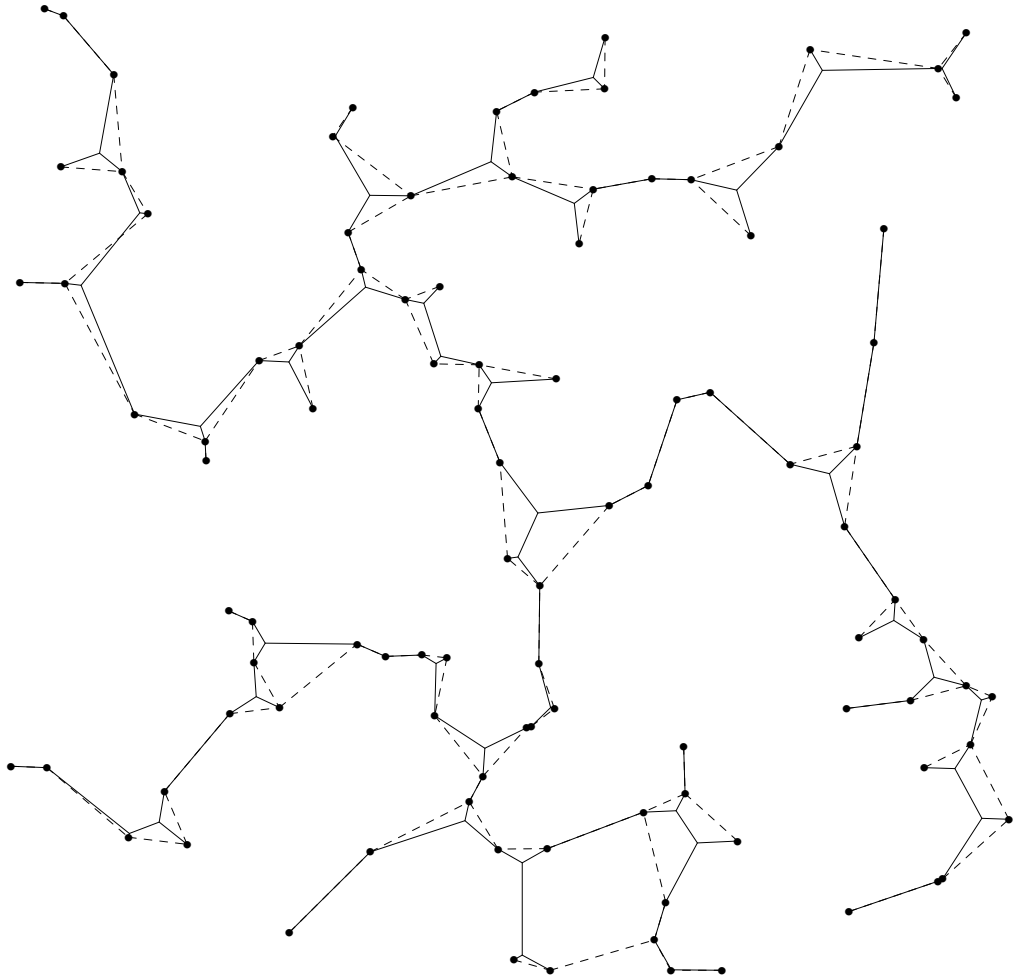


Figure B.18. Cockayne and Hewgill's Test Problem 18 Steiner minimal tree and minimum spanning tree.

B.19 Test Problem 19

Minimum Spanning Tree	6.6043
Steiner Minimal Tree	6.3689
Reduction	3.57%

Given Points

1	(0.09380000, 0.04090000)	51	(0.60479999, 0.26949999)
2	(0.13280000, 0.14420000)	52	(0.64469999, 0.40200001)
3	(0.09450000, 0.23639999)	53	(0.35890001, 0.19970000)
4	(0.08210000, 0.27039999)	54	(0.81930000, 0.54509997)
5	(0.07720000, 0.27440000)	55	(0.71829998, 0.09530000)
6	(0.01790000, 0.30669999)	56	(0.34119999, 0.73159999)
7	(0.04610000, 0.39969999)	57	(0.09840000, 0.72689998)
8	(0.01470000, 0.48320001)	58	(0.85869998, 0.39129999)
9	(0.00700000, 0.58590001)	59	(0.61129999, 0.64760000)
10	(0.00320000, 0.69029999)	60	(0.72039998, 0.81779999)
11	(0.00180000, 0.74680001)	61	(0.41659999, 0.24480000)
12	(0.00400000, 0.92119998)	62	(0.14620000, 0.69180000)
13	(0.18340001, 0.85460001)	63	(0.69169998, 0.81370002)
14	(0.19470000, 0.85759997)	64	(0.20130000, 0.11020000)
15	(0.30109999, 0.91339999)	65	(0.88029999, 0.48840001)
16	(0.33050001, 0.93049997)	66	(0.42910001, 0.27300000)
17	(0.56169999, 0.93559998)	67	(0.75239998, 0.29980001)
18	(0.65789998, 0.93019998)	68	(0.35859999, 0.58719999)
19	(0.69559997, 0.94290000)	69	(0.52620000, 0.40779999)
20	(0.70760000, 0.95130002)	70	(0.60509998, 0.47940001)
21	(0.76239997, 0.94190001)	71	(0.18290000, 0.67199999)
22	(0.82849997, 0.93879998)	72	(0.68250000, 0.41530001)
23	(0.97240001, 0.99070001)	73	(0.24230000, 0.17640001)
24	(0.99150002, 0.97189999)	74	(0.74959999, 0.41080001)
25	(0.97930002, 0.88120002)	75	(0.68260002, 0.89029998)
26	(0.98390001, 0.74070001)	76	(0.76539999, 0.53259999)
27	(0.98189998, 0.56080002)	77	(0.48429999, 0.35139999)
28	(0.97850001, 0.53149998)	78	(0.37979999, 0.28979999)
29	(0.96880001, 0.41520000)	79	(0.90259999, 0.47139999)
30	(0.98750001, 0.33120000)	80	(0.30350000, 0.47029999)
31	(0.95510000, 0.27010000)	81	(0.89020002, 0.29760000)
32	(0.94919997, 0.20660000)	82	(0.07740000, 0.36739999)
33	(0.87910002, 0.21070001)	83	(0.64190000, 0.67600000)
34	(0.78259999, 0.08580000)	84	(0.81250000, 0.38960001)
35	(0.71670002, 0.05710000)	85	(0.75629997, 0.13150001)
36	(0.57590002, 0.12460000)	86	(0.95490003, 0.39489999)
37	(0.39739999, 0.13090000)	87	(0.61290002, 0.86140001)
38	(0.28979999, 0.05740000)	88	(0.43290001, 0.20819999)
39	(0.21709999, 0.09880000)	89	(0.25940001, 0.12500000)
40	(0.93390000, 0.81290001)	90	(0.71710002, 0.46000001)
41	(0.47400001, 0.77170002)	91	(0.73379999, 0.31790000)
42	(0.59660000, 0.60329998)	92	(0.78549999, 0.41790000)
43	(0.54479998, 0.18979999)	93	(0.42590001, 0.46540001)
44	(0.90590000, 0.91939998)	94	(0.80089998, 0.32069999)
45	(0.73470002, 0.53799999)	95	(0.73809999, 0.45400000)
46	(0.34619999, 0.16910000)	96	(0.56070000, 0.39489999)
47	(0.54780000, 0.91090000)	97	(0.86350000, 0.67570001)
48	(0.53890002, 0.80440003)	98	(0.91759998, 0.31500000)
49	(0.34140000, 0.28140000)	99	(0.37509999, 0.67460001)
50	(0.13259999, 0.81610000)	100	(0.46970001, 0.77209997)

Optimal Steiner Points

1	(0.10386238, 0.81002718)	19	(0.35278931, 0.17122671)
2	(0.08251346, 0.74458581)	20	(0.26695853, 0.15028504)
3	(0.01634834, 0.73060387)	21	(0.25154573, 0.10435888)
4	(0.07170947, 0.36662549)	22	(0.49578086, 0.39884841)
5	(0.04892113, 0.31092206)	23	(0.60647953, 0.42284089)
6	(0.94726777, 0.75229347)	24	(0.72880834, 0.46319970)
7	(0.97384608, 0.88122964)	25	(0.74561489, 0.52719307)
8	(0.95222694, 0.91914874)	26	(0.75539660, 0.41962892)
9	(0.98309249, 0.97194558)	27	(0.81949860, 0.38213617)
10	(0.57427162, 0.86234498)	28	(0.96782517, 0.41524136)
11	(0.66695607, 0.86222410)	29	(0.93932074, 0.46984023)
12	(0.69375819, 0.81739384)	30	(0.94457579, 0.30802780)
13	(0.67528170, 0.92457044)	31	(0.97142267, 0.33535537)
14	(0.37412089, 0.71367729)	32	(0.93718630, 0.22092542)
15	(0.35929260, 0.50312763)	33	(0.70929945, 0.08658028)
16	(0.75282043, 0.30240521)	34	(0.75304180, 0.10913378)
17	(0.41726065, 0.26663393)	35	(0.54386210, 0.19214846)
18	(0.41379401, 0.22272550)	36	(0.14620182, 0.11520651)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	36	57.1%
3	20	31.7%
4	5	7.9%
5	2	3.2%

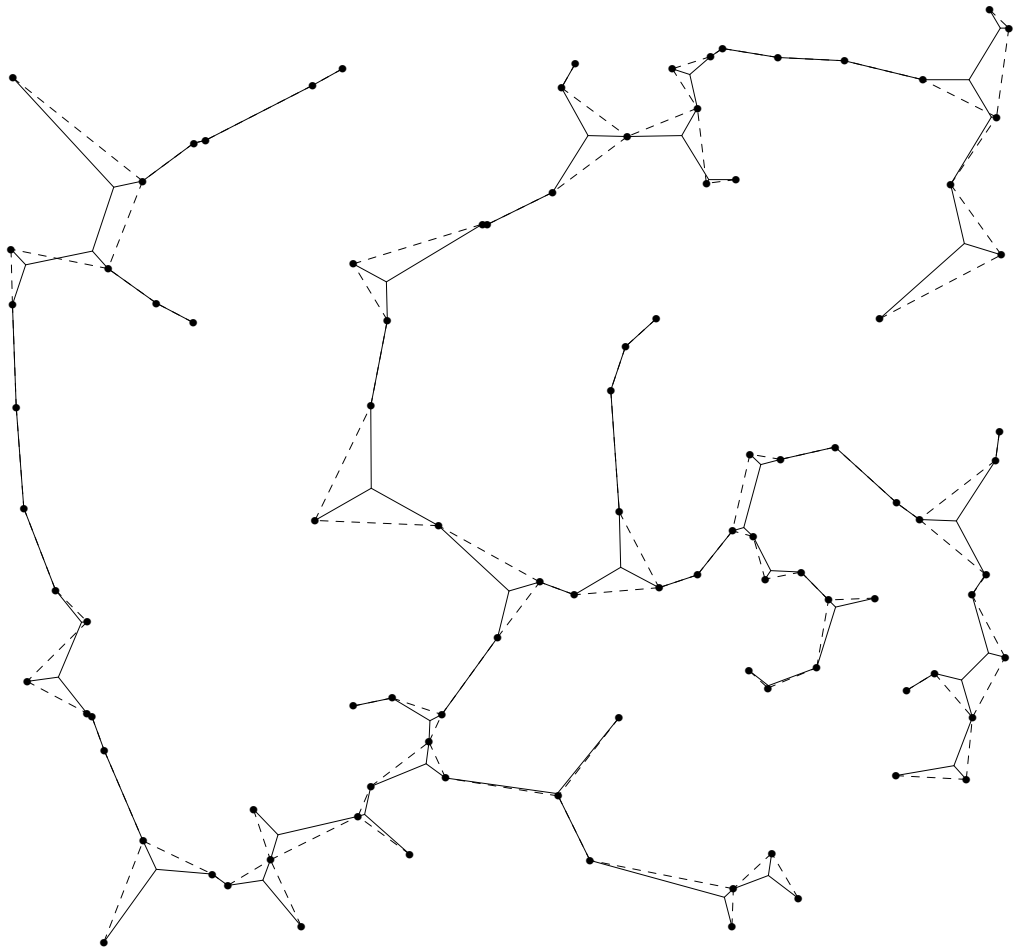


Figure B.19. Cockayne and Hewgill's Test Problem 19 Steiner minimal tree and minimum spanning tree.

B.20 Test Problem 20

Minimum Spanning Tree	6.9504
Steiner Minimal Tree	6.7482
Reduction	2.91%

Given Points

1	(0.15400000, 0.01480000)	51	(0.61680001, 0.25600001)
2	(0.05090000, 0.04940000)	52	(0.82130003, 0.59539998)
3	(0.02120000, 0.10790000)	53	(0.71980000, 0.67129999)
4	(0.00260000, 0.11400000)	54	(0.85979998, 0.32269999)
5	(0.00000000, 0.21310000)	55	(0.53090000, 0.63569999)
6	(0.01760000, 0.38310000)	56	(0.59009999, 0.61900002)
7	(0.03610000, 0.41389999)	57	(0.33059999, 0.28520000)
8	(0.10430000, 0.52359998)	58	(0.46689999, 0.38060001)
9	(0.08940000, 0.62180001)	59	(0.36050001, 0.29949999)
10	(0.02060000, 0.84369999)	60	(0.53839999, 0.26910001)
11	(0.00530000, 0.87610000)	61	(0.67479998, 0.61320001)
12	(0.20990001, 0.99229997)	62	(0.66549999, 0.11940000)
13	(0.48400000, 0.87419999)	63	(0.20110001, 0.18490000)
14	(0.56889999, 0.89819998)	64	(0.83530003, 0.06730000)
15	(0.69270003, 0.96149999)	65	(0.20960000, 0.82080001)
16	(0.70050001, 0.95400000)	66	(0.41710001, 0.40799999)
17	(0.92650002, 0.93229997)	67	(0.19820000, 0.90969998)
18	(0.99110001, 0.93279999)	68	(0.66200000, 0.67519999)
19	(0.94540000, 0.80210000)	69	(0.30100000, 0.49640000)
20	(0.91750002, 0.71060002)	70	(0.24560000, 0.71139997)
21	(0.92830002, 0.60089999)	71	(0.70190001, 0.42660001)
22	(0.99779999, 0.42359999)	72	(0.61699998, 0.47229999)
23	(0.97970003, 0.38970000)	73	(0.32650000, 0.77359998)
24	(0.96439999, 0.18260001)	74	(0.57459998, 0.17120001)
25	(0.93470001, 0.01640000)	75	(0.51650000, 0.12270000)
26	(0.82720000, 0.02380000)	76	(0.11940000, 0.52859998)
27	(0.81440002, 0.02350000)	77	(0.57470000, 0.27180001)
28	(0.73799998, 0.01760000)	78	(0.23610000, 0.12270000)
29	(0.67699999, 0.06330000)	79	(0.74910003, 0.76480001)
30	(0.55980003, 0.06290000)	80	(0.20890000, 0.45490000)
31	(0.52410001, 0.04580000)	81	(0.81489998, 0.52370000)
32	(0.46000001, 0.05760000)	82	(0.21460000, 0.42060000)
33	(0.45240000, 0.05100000)	83	(0.15030000, 0.82349998)
34	(0.39890000, 0.05910000)	84	(0.39629999, 0.75269997)
35	(0.32200000, 0.04970000)	85	(0.69900000, 0.40230000)
36	(0.26230001, 0.02940000)	86	(0.22450000, 0.67670000)
37	(0.08740000, 0.40070000)	87	(0.78439999, 0.24270000)
38	(0.35270000, 0.60089999)	88	(0.29290000, 0.64459997)
39	(0.74680001, 0.82510000)	89	(0.80800003, 0.16630000)
40	(0.33600000, 0.26730001)	90	(0.54030001, 0.33970001)
41	(0.17620000, 0.18220000)	91	(0.77010000, 0.26840001)
42	(0.56510001, 0.14229999)	92	(0.91140002, 0.25450000)
43	(0.28900000, 0.47589999)	93	(0.74379998, 0.70220000)
44	(0.41790000, 0.54490000)	94	(0.46410000, 0.45449999)
45	(0.35139999, 0.21799999)	95	(0.14270000, 0.41540000)
46	(0.22990000, 0.21760000)	96	(0.33809999, 0.75029999)
47	(0.48109999, 0.29139999)	97	(0.49450001, 0.77929997)
48	(0.82139999, 0.41450000)	98	(0.59119999, 0.57709998)
49	(0.89179999, 0.39700001)	99	(0.27710000, 0.34009999)
50	(0.80360001, 0.29560000)	100	(0.07280000, 0.21140000)

Optimal Steiner Points

1	(0.48277599, 0.78833091)	19	(0.77694100, 0.26684192)
2	(0.49328816, 0.86704540)	20	(0.84296119, 0.03990952)
3	(0.18760964, 0.83824867)	21	(0.53652430, 0.11630607)
4	(0.32412088, 0.76720899)	22	(0.54728979, 0.06689465)
5	(0.26250386, 0.75680786)	23	(0.57765269, 0.15257776)
6	(0.24192151, 0.68387622)	24	(0.59597468, 0.25045612)
7	(0.03817216, 0.40850064)	25	(0.23047601, 0.05127376)
8	(0.95320553, 0.91430098)	26	(0.20692532, 0.18442100)
9	(0.82579851, 0.59174430)	27	(0.02367943, 0.19335267)
10	(0.90319705, 0.62130040)	28	(0.01043397, 0.11687827)
11	(0.60038465, 0.60817516)	29	(0.51615328, 0.33283207)
12	(0.66483438, 0.62368900)	30	(0.50762701, 0.29894495)
13	(0.67577416, 0.66070241)	31	(0.44478402, 0.41094941)
14	(0.85630095, 0.30967984)	32	(0.36386335, 0.55934262)
15	(0.86491507, 0.38431764)	33	(0.21840273, 0.44746011)
16	(0.70506281, 0.42331409)	34	(0.16415876, 0.45411694)
17	(0.79215914, 0.44487882)	35	(0.11180294, 0.53080273)
18	(0.97805810, 0.39288542)	36	(0.33513159, 0.28290933)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	36	57.1%
3	19	30.2%
4	7	11.1%
5	1	1.6%

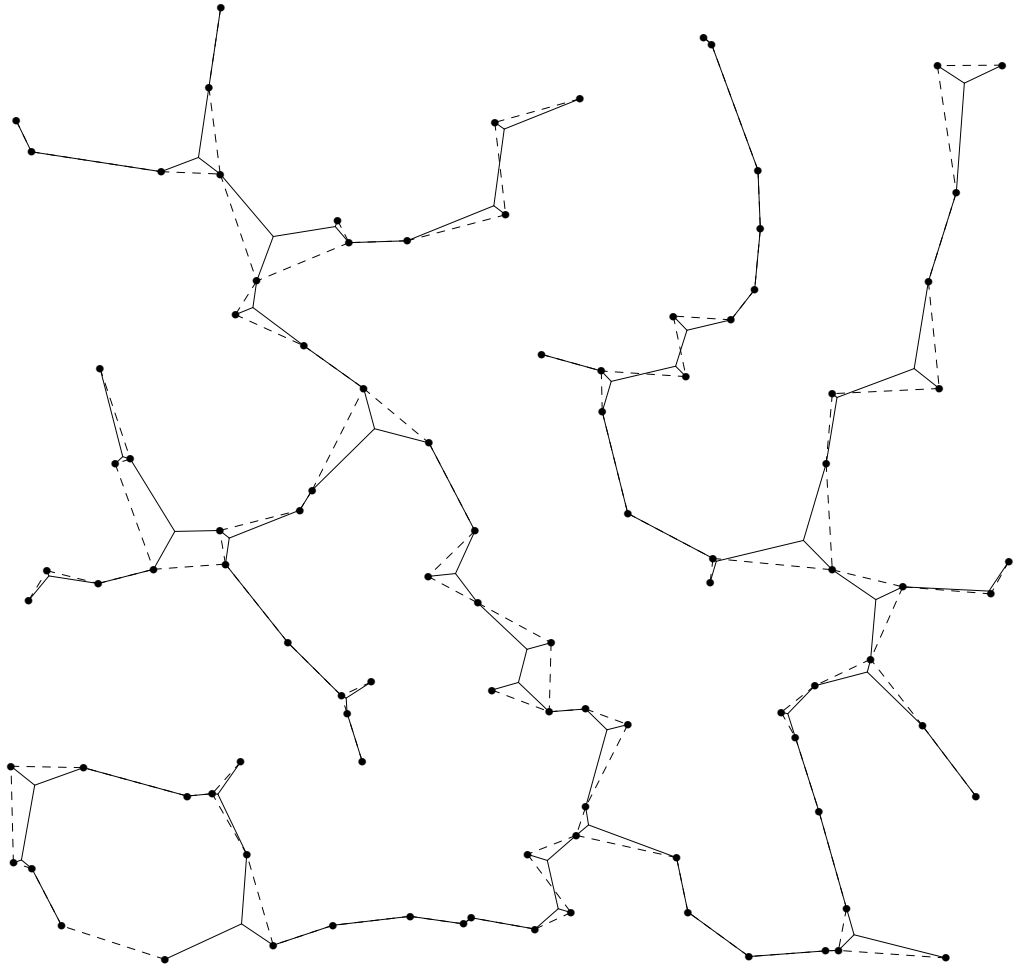


Figure B.20. Cockayne and Hewgill's Test Problem 20 Steiner minimal tree and minimum spanning tree.

B.21 Test Problem 21

Minimum Spanning Tree	6.5704
Steiner Minimal Tree	6.3276
Reduction	3.69%

Given Points

1	(0.87050003, 0.00010000)	51	(0.66869998, 0.19100000)
2	(0.82309997, 0.04710000)	52	(0.60259998, 0.44900000)
3	(0.70880002, 0.02020000)	53	(0.21990000, 0.83710003)
4	(0.67240000, 0.00630000)	54	(0.77200001, 0.42280000)
5	(0.63700002, 0.03480000)	55	(0.77600002, 0.49169999)
6	(0.58450001, 0.01920000)	56	(0.35900000, 0.44800001)
7	(0.48600000, 0.04230000)	57	(0.42340001, 0.85470003)
8	(0.30649999, 0.07170000)	58	(0.63139999, 0.69700003)
9	(0.15880001, 0.11590000)	59	(0.46370000, 0.79699999)
10	(0.01190000, 0.15889999)	60	(0.41729999, 0.20530000)
11	(0.00000000, 0.21830000)	61	(0.77460003, 0.22640000)
12	(0.05600000, 0.50849998)	62	(0.21940000, 0.94360000)
13	(0.12540001, 0.60650003)	63	(0.52300000, 0.75379997)
14	(0.13480000, 0.76950002)	64	(0.84710002, 0.61080003)
15	(0.08970000, 0.83550000)	65	(0.45649999, 0.44119999)
16	(0.00840000, 0.97850001)	66	(0.54240000, 0.82340002)
17	(0.17340000, 0.99680001)	67	(0.67930001, 0.16630000)
18	(0.25000000, 0.97790003)	68	(0.69370002, 0.50940001)
19	(0.44409999, 0.96039999)	69	(0.66579998, 0.78009999)
20	(0.64150000, 0.98930001)	70	(0.83810002, 0.16710000)
21	(0.68919998, 0.98089999)	71	(0.50720000, 0.83380002)
22	(0.72030002, 0.94069999)	72	(0.76779997, 0.33550000)
23	(0.77880001, 0.82720000)	73	(0.78960001, 0.56029999)
24	(0.91310000, 0.81389999)	74	(0.08400000, 0.50900000)
25	(0.90640002, 0.65240002)	75	(0.82059997, 0.25340000)
26	(0.95709997, 0.42870000)	76	(0.71660000, 0.35659999)
27	(0.98500001, 0.32460001)	77	(0.67869997, 0.20750000)
28	(0.92760003, 0.01100000)	78	(0.42510000, 0.23350000)
29	(0.30669999, 0.85619998)	79	(0.21089999, 0.69910002)
30	(0.05310000, 0.19599999)	80	(0.51209998, 0.70330000)
31	(0.03460000, 0.18290000)	81	(0.55930001, 0.74739999)
32	(0.65820003, 0.11630000)	82	(0.20960000, 0.57580000)
33	(0.42469999, 0.41890001)	83	(0.66570002, 0.94120002)
34	(0.44060001, 0.57660002)	84	(0.36759999, 0.83170003)
35	(0.18380000, 0.90730000)	85	(0.72000003, 0.71569997)
36	(0.62440002, 0.80599999)	86	(0.38150001, 0.32440001)
37	(0.27649999, 0.59990001)	87	(0.54479998, 0.53240001)
38	(0.84189999, 0.21600001)	88	(0.21150000, 0.73989999)
39	(0.60030001, 0.91360003)	89	(0.47940001, 0.29060000)
40	(0.21740000, 0.43040001)	90	(0.63040000, 0.50000000)
41	(0.51090002, 0.14410000)	91	(0.45580000, 0.47799999)
42	(0.85219997, 0.47350001)	92	(0.21840000, 0.65829998)
43	(0.59140003, 0.52420002)	93	(0.82620001, 0.14420000)
44	(0.18940000, 0.57910001)	94	(0.22370000, 0.88840002)
45	(0.36460000, 0.65149999)	95	(0.19110000, 0.74529999)
46	(0.30970001, 0.36570001)	96	(0.19520000, 0.36790001)
47	(0.67449999, 0.04030000)	97	(0.65660000, 0.93989998)
48	(0.49890000, 0.06530000)	98	(0.70359999, 0.18410000)
49	(0.27180001, 0.33890000)	99	(0.22390001, 0.36300001)
50	(0.32679999, 0.62230003)	100	(0.30770001, 0.09500000)

Optimal Steiner Points

1	(0.50050420, 0.05353669)	19	(0.58622378, 0.83878511)
2	(0.65961027, 0.04848860)	20	(0.68796140, 0.73260111)
3	(0.68322301, 0.02214162)	21	(0.68656564, 0.76914561)
4	(0.68394154, 0.18012255)	22	(0.53006101, 0.74418128)
5	(0.67348439, 0.19196799)	23	(0.17833160, 0.76856053)
6	(0.20979050, 0.37389311)	24	(0.23659997, 0.86153418)
7	(0.05264945, 0.19118457)	25	(0.20622537, 0.91055655)
8	(0.02365880, 0.18227082)	26	(0.22101210, 0.96407473)
9	(0.30113667, 0.08752259)	27	(0.20782027, 0.73736131)
10	(0.43442035, 0.27624550)	28	(0.13609335, 0.58367103)
11	(0.35369825, 0.36775303)	29	(0.23648249, 0.60730052)
12	(0.37902808, 0.41676891)	30	(0.36262310, 0.63696080)
13	(0.27172738, 0.33988684)	31	(0.47727200, 0.52562326)
14	(0.68331718, 0.95523530)	32	(0.61705816, 0.49687198)
15	(0.68050253, 0.97397083)	33	(0.79278666, 0.46798998)
16	(0.42006719, 0.84380466)	34	(0.76358968, 0.51166928)
17	(0.51322085, 0.82338446)	35	(0.74751949, 0.36113647)
18	(0.49801710, 0.79701835)	36	(0.79874671, 0.25668991)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	32	50.8%
3	26	41.3%
4	5	7.9%

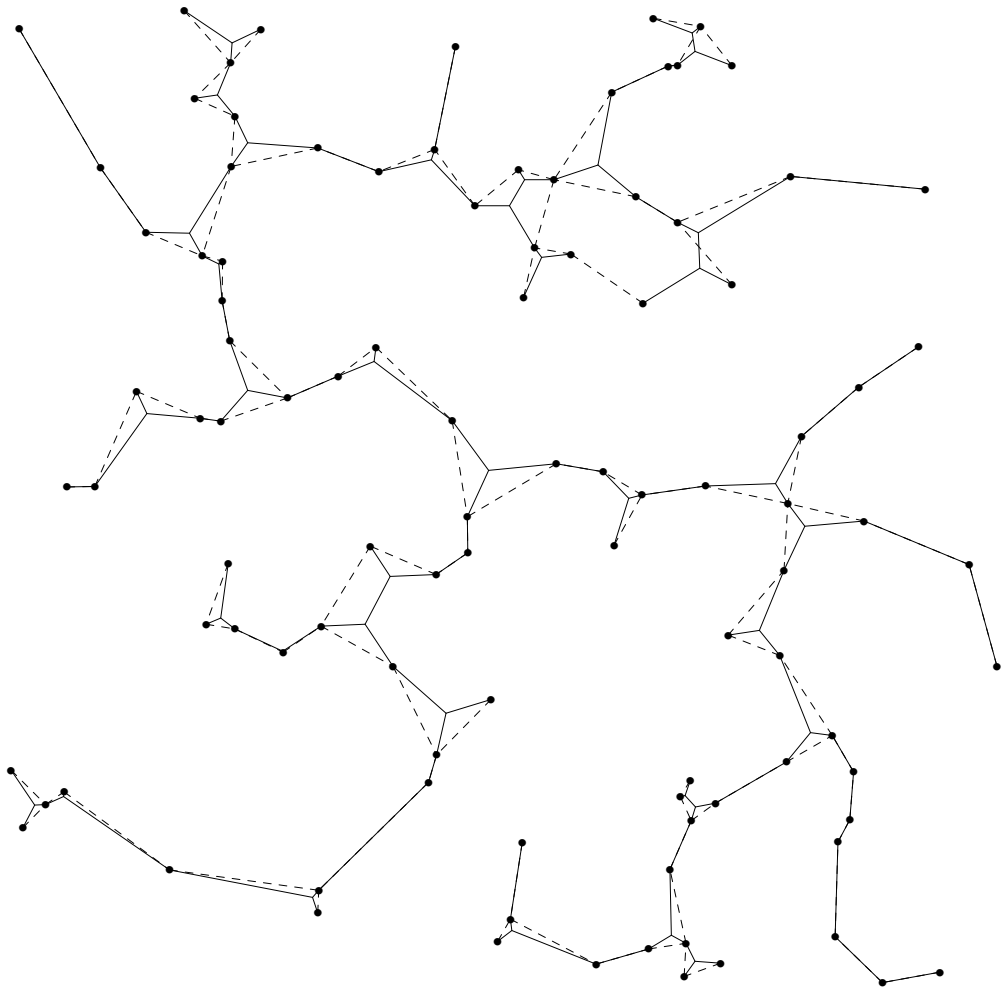


Figure B.21. Cockayne and Hewgill's Test Problem 21 Steiner minimal tree and minimum spanning tree.

B.22 Test Problem 22

Minimum Spanning Tree	6.7205
Steiner Minimal Tree	6.5233
Reduction	2.93%

Given Points

1	(0.82650000, 0.00690000)	51	(0.89099997, 0.25569999)
2	(0.72880000, 0.01740000)	52	(0.71259999, 0.73869997)
3	(0.69859999, 0.01570000)	53	(0.50790000, 0.69970000)
4	(0.57330000, 0.03390000)	54	(0.60420001, 0.57359999)
5	(0.52340001, 0.02380000)	55	(0.66299999, 0.13750000)
6	(0.33590001, 0.01380000)	56	(0.52670002, 0.45359999)
7	(0.21160001, 0.02260000)	57	(0.43560001, 0.39910001)
8	(0.14579999, 0.05880000)	58	(0.13820000, 0.35560000)
9	(0.00000000, 0.21340001)	59	(0.70999998, 0.17090000)
10	(0.01970000, 0.54130000)	60	(0.91920000, 0.10480000)
11	(0.04490000, 0.61540002)	61	(0.86500001, 0.11900000)
12	(0.01760000, 0.69610000)	62	(0.14320000, 0.16949999)
13	(0.03060000, 0.71359998)	63	(0.74860001, 0.04940000)
14	(0.11040000, 0.82510000)	64	(0.09070000, 0.72140002)
15	(0.16270000, 0.91240001)	65	(0.48890001, 0.10770000)
16	(0.33039999, 0.89480001)	66	(0.82639998, 0.62699997)
17	(0.39629999, 0.92229998)	67	(0.61339998, 0.60659999)
18	(0.39989999, 0.93360001)	68	(0.42390001, 0.57520002)
19	(0.53649998, 0.92940003)	69	(0.56639999, 0.08750000)
20	(0.77569997, 0.94110000)	70	(0.52850002, 0.87080002)
21	(0.85110003, 0.94090003)	71	(0.18990000, 0.78119999)
22	(0.93099999, 0.96350002)	72	(0.37239999, 0.35789999)
23	(0.89740002, 0.88290000)	73	(0.43419999, 0.55150002)
24	(0.95029998, 0.76810002)	74	(0.50269997, 0.44510001)
25	(0.94889998, 0.61619997)	75	(0.31360000, 0.85619998)
26	(0.97530001, 0.40759999)	76	(0.26989999, 0.19810000)
27	(0.96939999, 0.31009999)	77	(0.16670001, 0.42950001)
28	(0.97289997, 0.06280000)	78	(0.50300002, 0.75059998)
29	(0.96259999, 0.04610000)	79	(0.11280000, 0.30039999)
30	(0.50889999, 0.78259999)	80	(0.61330003, 0.37560001)
31	(0.79820001, 0.12340000)	81	(0.63590002, 0.36750001)
32	(0.83870000, 0.21520001)	82	(0.54420000, 0.13869999)
33	(0.41389999, 0.25080001)	83	(0.36809999, 0.49840000)
34	(0.74860001, 0.24940000)	84	(0.13940001, 0.40120000)
35	(0.91000003, 0.31410000)	85	(0.49950001, 0.75300002)
36	(0.20080000, 0.54180002)	86	(0.23350000, 0.25619999)
37	(0.86119998, 0.85549998)	87	(0.33820000, 0.48199999)
38	(0.22200000, 0.52240002)	88	(0.77039999, 0.13920000)
39	(0.49770001, 0.08140000)	89	(0.64700001, 0.82819998)
40	(0.76480001, 0.42940000)	90	(0.37090001, 0.06350000)
41	(0.11320000, 0.47299999)	91	(0.66949999, 0.84050000)
42	(0.20670000, 0.23690000)	92	(0.38949999, 0.08330000)
43	(0.16869999, 0.55010003)	93	(0.28220001, 0.63679999)
44	(0.83209997, 0.61430001)	94	(0.51959997, 0.24710000)
45	(0.47330001, 0.47549999)	95	(0.41020000, 0.75269997)
46	(0.41610000, 0.91259998)	96	(0.57260001, 0.81840003)
47	(0.47839999, 0.50790000)	97	(0.23810001, 0.50360000)
48	(0.52759999, 0.66579998)	98	(0.55040002, 0.62540001)
49	(0.13160001, 0.23830000)	99	(0.91180003, 0.44250000)
50	(0.48879999, 0.08500000)	100	(0.16580001, 0.14110000)

Optimal Steiner Points

1	(0.15670282, 0.06349860)	21	(0.43334818, 0.53230667)
2	(0.22913000, 0.24202374)	22	(0.47465143, 0.50595200)
3	(0.10698643, 0.25731272)	23	(0.43290782, 0.57077456)
4	(0.15331289, 0.21906909)	24	(0.52962416, 0.63560706)
5	(0.13133049, 0.78417844)	25	(0.60259628, 0.59943509)
6	(0.11297057, 0.51754069)	26	(0.49639985, 0.75771445)
7	(0.04520500, 0.55620104)	27	(0.54469860, 0.82381272)
8	(0.14962688, 0.42698854)	28	(0.66215342, 0.82805395)
9	(0.21369836, 0.54057050)	29	(0.88904500, 0.92753857)
10	(0.39928046, 0.92377889)	30	(0.88605654, 0.85792518)
11	(0.51542622, 0.90105504)	31	(0.83354706, 0.62009031)
12	(0.96604568, 0.06073669)	32	(0.91945142, 0.64467043)
13	(0.48261690, 0.09344303)	33	(0.52657056, 0.45297679)
14	(0.53030998, 0.04660446)	34	(0.94776964, 0.32806411)
15	(0.55581272, 0.05255927)	35	(0.96043867, 0.40210551)
16	(0.57237875, 0.11149763)	36	(0.80874777, 0.11028455)
17	(0.51272041, 0.23963921)	37	(0.78747636, 0.05542334)
18	(0.42235488, 0.25996923)	38	(0.74976259, 0.17226686)
19	(0.39327842, 0.35320291)	39	(0.77138865, 0.21288612)
20	(0.48155969, 0.44969139)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	31	51.7%
3	20	33.3%
4	8	13.3%
5	1	1.7%

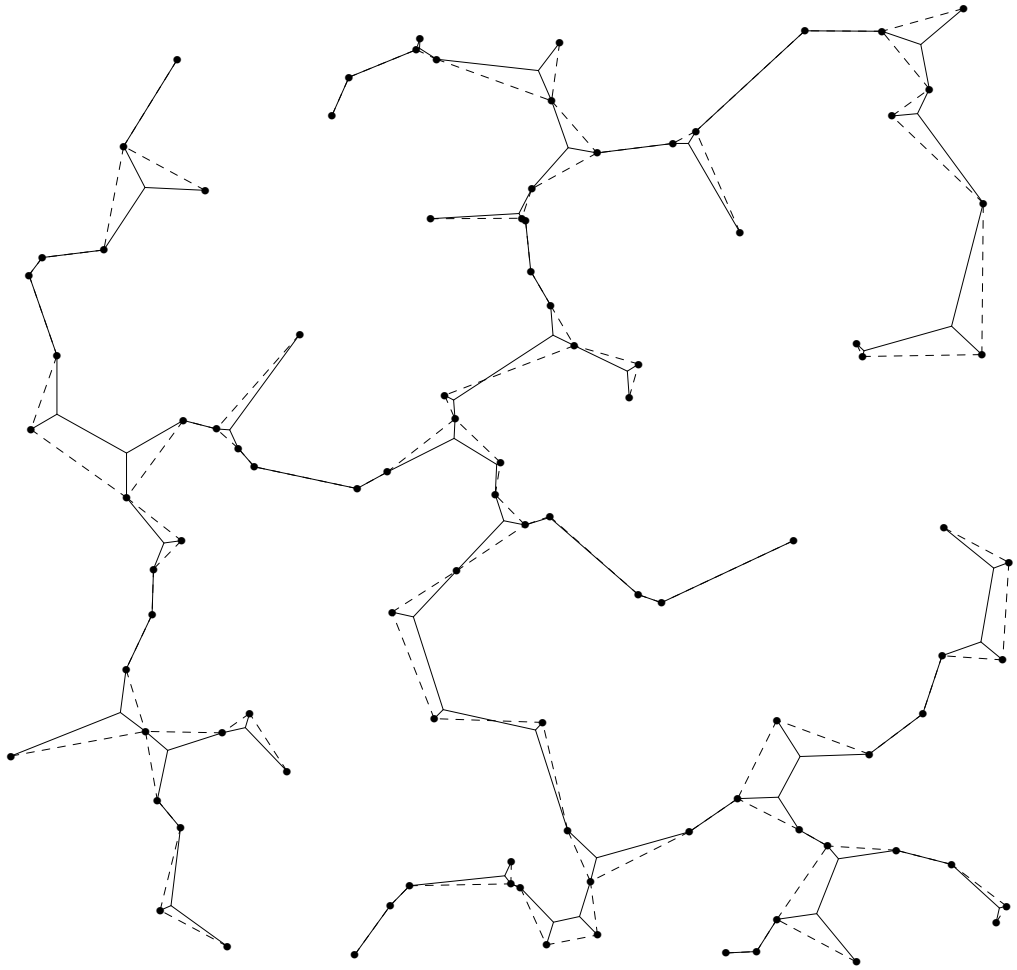


Figure B.22. Cockayne and Hewgill's Test Problem 22 Steiner minimal tree and minimum spanning tree.

B.23 Test Problem 23

Minimum Spanning Tree	6.9269
Steiner Minimal Tree	6.6861
Reduction	3.48%

Given Points

1	(0.85699999, 0.01250000)	51	(0.75569999, 0.22700000)
2	(0.81999999, 0.01800000)	52	(0.33160001, 0.49849999)
3	(0.81089997, 0.02780000)	53	(0.60890001, 0.26879999)
4	(0.74760002, 0.04300000)	54	(0.32240000, 0.24270000)
5	(0.58289999, 0.13730000)	55	(0.71910000, 0.62730002)
6	(0.46070001, 0.06840000)	56	(0.65289998, 0.31709999)
7	(0.27039999, 0.09660000)	57	(0.78390002, 0.73710001)
8	(0.14630000, 0.10800000)	58	(0.91880000, 0.86180001)
9	(0.04940000, 0.02110000)	59	(0.91270000, 0.29620001)
10	(0.02990000, 0.17850000)	60	(0.84329998, 0.88000000)
11	(0.00000000, 0.21349999)	61	(0.18960001, 0.96230000)
12	(0.12170000, 0.39420000)	62	(0.73449999, 0.40079999)
13	(0.12989999, 0.60540003)	63	(0.15660000, 0.29409999)
14	(0.07660000, 0.66360003)	64	(0.77200001, 0.44729999)
15	(0.05320000, 0.72899997)	65	(0.13540000, 0.17860000)
16	(0.05940000, 0.74070001)	66	(0.29690000, 0.14480001)
17	(0.04210000, 0.91339999)	67	(0.70069999, 0.25569999)
18	(0.02400000, 0.98430002)	68	(0.48310000, 0.13740000)
19	(0.08600000, 0.94999999)	69	(0.40560001, 0.34470001)
20	(0.17490000, 0.99930000)	70	(0.13200000, 0.72189999)
21	(0.39610001, 0.96899998)	71	(0.71050000, 0.92670000)
22	(0.55690002, 0.98760003)	72	(0.79920000, 0.31790000)
23	(0.79229999, 0.99269998)	73	(0.61250001, 0.49649999)
24	(0.95469999, 0.97479999)	74	(0.80390000, 0.88770002)
25	(0.96100003, 0.91970003)	75	(0.39340001, 0.77880001)
26	(0.97119999, 0.89029998)	76	(0.58810002, 0.54600000)
27	(0.99059999, 0.80019999)	77	(0.79689997, 0.70620000)
28	(0.97759998, 0.79600000)	78	(0.92919999, 0.23989999)
29	(0.96929997, 0.76310003)	79	(0.59680003, 0.33950001)
30	(0.96109998, 0.56190002)	80	(0.49059999, 0.49910000)
31	(0.99959999, 0.44440001)	81	(0.07820000, 0.14810000)
32	(0.96149999, 0.22300000)	82	(0.28639999, 0.34570000)
33	(0.96069998, 0.09640000)	83	(0.36919999, 0.84079999)
34	(0.90619999, 0.02180000)	84	(0.57870001, 0.58789998)
35	(0.71950001, 0.25900000)	85	(0.35900000, 0.41589999)
36	(0.77780002, 0.82249999)	86	(0.19480000, 0.61659998)
37	(0.70209998, 0.59230000)	87	(0.36440000, 0.15490000)
38	(0.40220001, 0.65030003)	88	(0.32679999, 0.16710000)
39	(0.78320003, 0.55739999)	89	(0.21870001, 0.39410001)
40	(0.47080001, 0.18500000)	90	(0.28279999, 0.71130002)
41	(0.15290000, 0.18930000)	91	(0.59230000, 0.63150001)
42	(0.42559999, 0.74269998)	92	(0.66750002, 0.64579999)
43	(0.16570000, 0.62900001)	93	(0.69660002, 0.49919999)
44	(0.57779998, 0.18340001)	94	(0.86949998, 0.02840000)
45	(0.59299999, 0.85909998)	95	(0.13600001, 0.92129999)
46	(0.64440000, 0.67360002)	96	(0.63370001, 0.78890002)
47	(0.62699997, 0.22409999)	97	(0.88110000, 0.48080000)
48	(0.63690001, 0.63319999)	98	(0.76349998, 0.65149999)
49	(0.46980000, 0.17309999)	99	(0.47389999, 0.54229999)
50	(0.21699999, 0.60100001)	100	(0.67960000, 0.67850000)

Optimal Steiner Points

1	(0.94466436, 0.49841136)	23	(0.21106468, 0.61471730)
2	(0.10797107, 0.14379032)	24	(0.49459803, 0.52583241)
3	(0.12078483, 0.11169360)	25	(0.57409942, 0.55713904)
4	(0.36307463, 0.36492014)	26	(0.61491597, 0.91329628)
5	(0.31526455, 0.33197543)	27	(0.76812565, 0.93429995)
6	(0.21857710, 0.39259201)	28	(0.80900294, 0.87927675)
7	(0.16246162, 0.36601788)	29	(0.70437527, 0.62032145)
8	(0.33046186, 0.16979086)	30	(0.75338030, 0.44504908)
9	(0.46761757, 0.17558154)	31	(0.71139091, 0.50098801)
10	(0.48798916, 0.14884263)	32	(0.73286450, 0.55131501)
11	(0.56710362, 0.15836625)	33	(0.77758300, 0.31876537)
12	(0.98075336, 0.80078375)	34	(0.73931074, 0.25820693)
13	(0.95114911, 0.85988241)	35	(0.91121536, 0.29453829)
14	(0.05479221, 0.94540823)	36	(0.63288361, 0.24178343)
15	(0.05745077, 0.73048031)	37	(0.62261093, 0.31072283)
16	(0.08596595, 0.70586795)	38	(0.66104800, 0.64904827)
17	(0.07799683, 0.66408652)	39	(0.66009462, 0.66569859)
18	(0.13217431, 0.61732417)	40	(0.59520119, 0.62771773)
19	(0.17972025, 0.96623290)	41	(0.96014875, 0.22221202)
20	(0.13463885, 0.93063796)	42	(0.85697246, 0.01256932)
21	(0.40906191, 0.74431419)	43	(0.90536505, 0.02379720)
22	(0.37874979, 0.70193499)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	28	50.0%
3	16	28.6%
4	10	17.9%
5	1	1.8%
6	1	1.8%

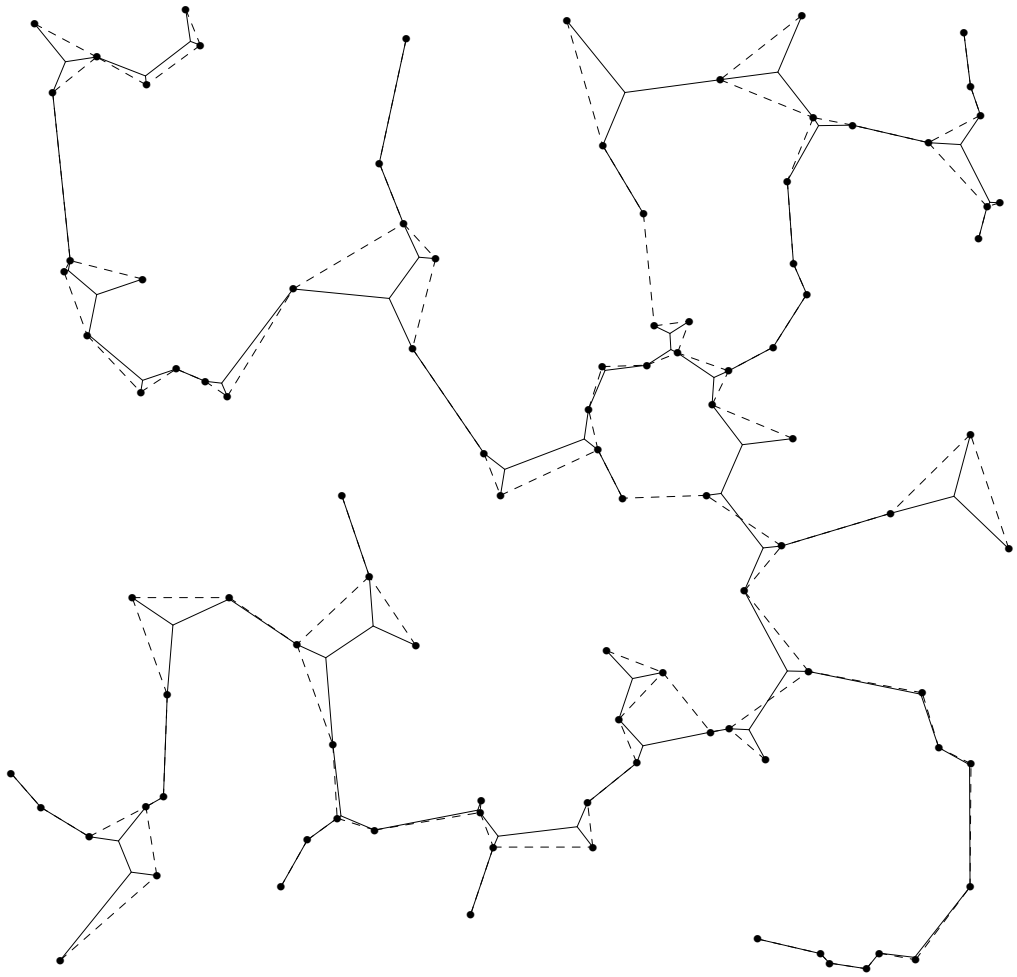


Figure B.23. Cockayne and Hewgill's Test Problem 23 Steiner minimal tree and minimum spanning tree.

B.24 Test Problem 24

Minimum Spanning Tree	6.6309
Steiner Minimal Tree	6.4394
Reduction	2.89%

Given Points

1	(0.80409998, 0.00260000)	51	(0.30550000, 0.42469999)
2	(0.78149998, 0.02280000)	52	(0.75900000, 0.71120000)
3	(0.51169997, 0.06600000)	53	(0.62910002, 0.80549997)
4	(0.44839999, 0.06890000)	54	(0.46410000, 0.84439999)
5	(0.41769999, 0.02620000)	55	(0.05540000, 0.68210000)
6	(0.33090001, 0.02580000)	56	(0.47060001, 0.84380001)
7	(0.07220000, 0.06690000)	57	(0.31690001, 0.69919997)
8	(0.05980000, 0.09790000)	58	(0.74169999, 0.31209999)
9	(0.00000000, 0.21840000)	59	(0.40640000, 0.79259998)
10	(0.06210000, 0.32339999)	60	(0.60189998, 0.14530000)
11	(0.05440000, 0.35550001)	61	(0.82419997, 0.88749999)
12	(0.10360000, 0.46689999)	62	(0.89029998, 0.79369998)
13	(0.00450000, 0.69440001)	63	(0.60390002, 0.15590000)
14	(0.02020000, 0.90850002)	64	(0.93460000, 0.18870001)
15	(0.14860000, 0.85740000)	65	(0.46959999, 0.38820001)
16	(0.22460000, 0.84660000)	66	(0.07690000, 0.78310001)
17	(0.30260000, 0.83490002)	67	(0.27640000, 0.38540000)
18	(0.34999999, 0.84179997)	68	(0.40799999, 0.22950000)
19	(0.36600000, 0.87639999)	69	(0.85769999, 0.73280001)
20	(0.39910001, 0.96689999)	70	(0.60219997, 0.30570000)
21	(0.71020001, 0.87879997)	71	(0.58700001, 0.22579999)
22	(0.80409998, 0.93120003)	72	(0.33939999, 0.54900002)
23	(0.83099997, 0.96810001)	73	(0.59770000, 0.81300002)
24	(0.94180000, 0.98290002)	74	(0.73689997, 0.35380000)
25	(0.99110001, 0.73600000)	75	(0.25340000, 0.66109997)
26	(0.97229999, 0.57529998)	76	(0.38499999, 0.56169999)
27	(0.95599997, 0.45390001)	77	(0.84939998, 0.44830000)
28	(0.95169997, 0.42320001)	78	(0.75120002, 0.63309997)
29	(0.97000003, 0.38540000)	79	(0.51580000, 0.29420000)
30	(0.91960001, 0.31400001)	80	(0.83829999, 0.81339997)
31	(0.95300001, 0.20479999)	81	(0.07020000, 0.24390000)
32	(0.95260000, 0.18220000)	82	(0.85650003, 0.47839999)
33	(0.88709998, 0.00750000)	83	(0.48789999, 0.21640000)
34	(0.84320003, 0.02050000)	84	(0.21300000, 0.43910000)
35	(0.83010000, 0.01410000)	85	(0.47799999, 0.62830001)
36	(0.51760000, 0.76169997)	86	(0.55909997, 0.35800001)
37	(0.50239998, 0.52910000)	87	(0.88239998, 0.46419999)
38	(0.79589999, 0.23400000)	88	(0.21580000, 0.24730000)
39	(0.31600001, 0.57239997)	89	(0.70220000, 0.12300000)
40	(0.46180001, 0.81699997)	90	(0.42080000, 0.84890002)
41	(0.81410003, 0.60039997)	91	(0.67189997, 0.47150001)
42	(0.43259999, 0.15770000)	92	(0.79990000, 0.45649999)
43	(0.80650002, 0.25700000)	93	(0.86979997, 0.20330000)
44	(0.53390002, 0.72229999)	94	(0.20680000, 0.22030000)
45	(0.80540001, 0.77649999)	95	(0.21580000, 0.25860000)
46	(0.58319998, 0.24879999)	96	(0.37439999, 0.39340001)
47	(0.36300001, 0.43610001)	97	(0.51249999, 0.29730001)
48	(0.87339997, 0.14170000)	98	(0.25299999, 0.64719999)
49	(0.77130002, 0.31920001)	99	(0.38699999, 0.28240001)
50	(0.91420001, 0.23019999)	100	(0.88690001, 0.48730001)

Optimal Steiner Points

1	(0.20274223, 0.23281811)	21	(0.80646080, 0.24006471)
2	(0.08319744, 0.25831497)	22	(0.91099554, 0.20706666)
3	(0.03310361, 0.20862892)	23	(0.88075238, 0.19477789)
4	(0.11424753, 0.44540218)	24	(0.85998261, 0.02772284)
5	(0.27564025, 0.40595695)	25	(0.80436283, 0.00424915)
6	(0.04217515, 0.69934708)	26	(0.96940160, 0.38545102)
7	(0.09101151, 0.84064543)	27	(0.93118691, 0.49050128)
8	(0.51896501, 0.77664840)	28	(0.87779951, 0.47308636)
9	(0.43041074, 0.83060873)	29	(0.85879910, 0.47395912)
10	(0.45516783, 0.82962245)	30	(0.84625727, 0.45437059)
11	(0.46428302, 0.84405166)	31	(0.94668746, 0.19012520)
12	(0.36995983, 0.86208427)	32	(0.60570794, 0.14909068)
13	(0.79967046, 0.90906012)	33	(0.51524675, 0.29409635)
14	(0.85997307, 0.77816010)	34	(0.44109711, 0.20049466)
15	(0.83726919, 0.79283029)	35	(0.45717213, 0.07837685)
16	(0.75664294, 0.63601899)	36	(0.34947962, 0.54272789)
17	(0.52111006, 0.34443635)	37	(0.34629980, 0.44649193)
18	(0.67607731, 0.38510150)	38	(0.37874621, 0.39918932)
19	(0.60070658, 0.33658779)	39	(0.46054259, 0.57677257)
20	(0.74963576, 0.32225126)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	30	50.0%
3	23	38.3%
4	5	8.3%
5	2	3.3%

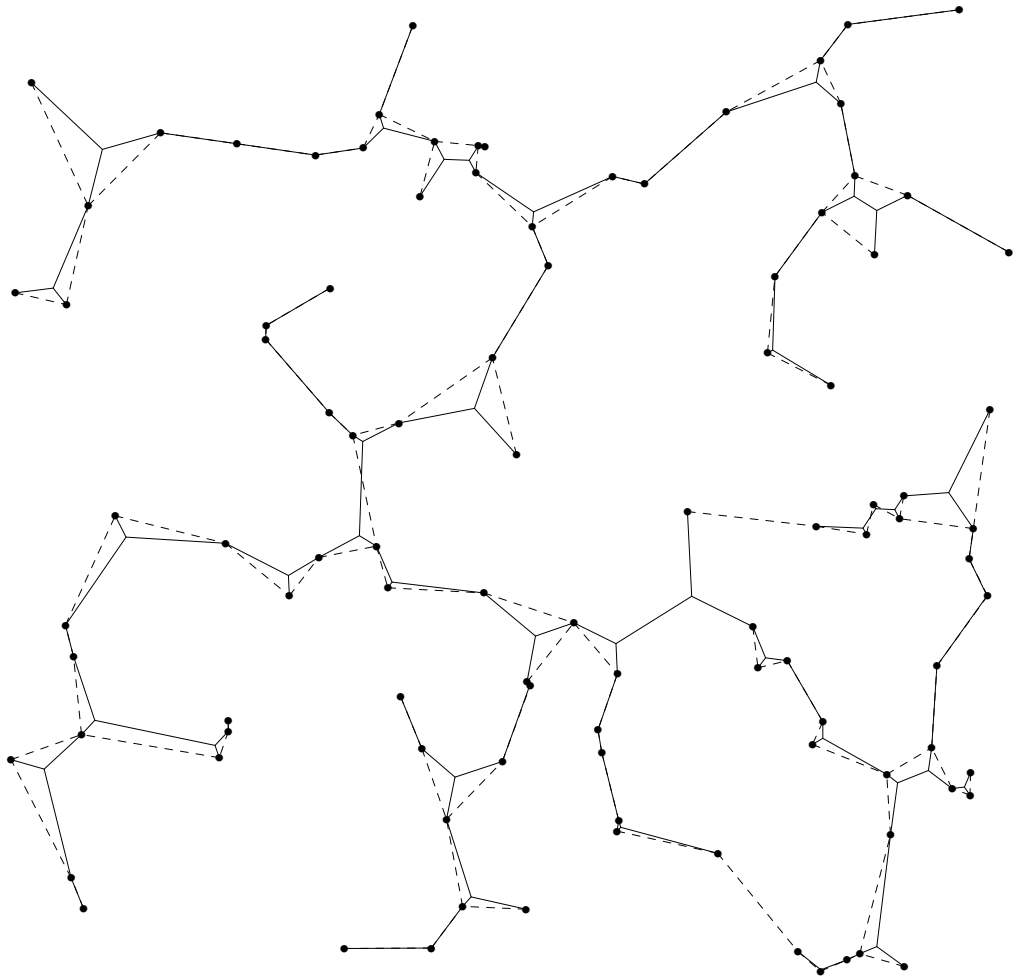


Figure B.24. Cockayne and Hewgill's Test Problem 24 Steiner minimal tree and minimum spanning tree.

B.25 Test Problem 25

Minimum Spanning Tree	6.7339
Steiner Minimal Tree	6.5265
Reduction	3.08%

Given Points

1	(0.34729999, 0.00550000)	51	(0.42890000, 0.29760000)
2	(0.23040000, 0.09420000)	52	(0.58630002, 0.37259999)
3	(0.20469999, 0.09090000)	53	(0.53259999, 0.11380000)
4	(0.14120001, 0.07110000)	54	(0.82499999, 0.33090001)
5	(0.10880000, 0.04450000)	55	(0.66990000, 0.46640000)
6	(0.05980000, 0.01850000)	56	(0.63900000, 0.42010000)
7	(0.01790000, 0.15920000)	57	(0.36840001, 0.52010000)
8	(0.00000000, 0.21380000)	58	(0.82980001, 0.76840001)
9	(0.01070000, 0.41960001)	59	(0.26550001, 0.47510001)
10	(0.04410000, 0.69430000)	60	(0.21330000, 0.52310002)
11	(0.03730000, 0.70670003)	61	(0.57969999, 0.52069998)
12	(0.03250000, 0.71280003)	62	(0.91060001, 0.21060000)
13	(0.01950000, 0.84219998)	63	(0.83039999, 0.11600000)
14	(0.12180000, 0.97070003)	64	(0.13530000, 0.45640001)
15	(0.12080000, 0.99910003)	65	(0.08540000, 0.85170001)
16	(0.37970001, 0.95560002)	66	(0.14870000, 0.22770000)
17	(0.59200001, 0.98610002)	67	(0.65350002, 0.91930002)
18	(0.64039999, 0.99680001)	68	(0.10680000, 0.49239999)
19	(0.71560001, 0.98299998)	69	(0.71630001, 0.69980001)
20	(0.76679999, 0.97630000)	70	(0.84700000, 0.80290002)
21	(0.87010002, 0.93769997)	71	(0.17870000, 0.70940000)
22	(0.93409997, 0.84719998)	72	(0.49370000, 0.12760000)
23	(0.92930001, 0.75849998)	73	(0.27270001, 0.58990002)
24	(0.97140002, 0.62550002)	74	(0.86989999, 0.82569999)
25	(0.92390001, 0.26600000)	75	(0.18030000, 0.72250003)
26	(0.98979998, 0.17000000)	76	(0.02770000, 0.22400001)
27	(0.90950000, 0.07570000)	77	(0.75739998, 0.74699998)
28	(0.89459997, 0.04010000)	78	(0.52219999, 0.68129998)
29	(0.77579999, 0.08440000)	79	(0.80199999, 0.69450003)
30	(0.68030000, 0.05290000)	80	(0.45950001, 0.28780001)
31	(0.62980002, 0.02620000)	81	(0.81080002, 0.38229999)
32	(0.53359997, 0.06920000)	82	(0.11460000, 0.25090000)
33	(0.39520001, 0.55110002)	83	(0.61470002, 0.73540002)
34	(0.68460000, 0.38949999)	84	(0.57380003, 0.77740002)
35	(0.29980001, 0.19069999)	85	(0.35940000, 0.38000000)
36	(0.73350000, 0.69660002)	86	(0.23610000, 0.11050000)
37	(0.19790000, 0.40869999)	87	(0.62430000, 0.25960001)
38	(0.20130000, 0.10960000)	88	(0.14480001, 0.46340001)
39	(0.59810001, 0.65759999)	89	(0.69029999, 0.43279999)
40	(0.21670000, 0.23510000)	90	(0.11280000, 0.78170002)
41	(0.90820003, 0.60159999)	91	(0.81150001, 0.67650002)
42	(0.06160000, 0.15480000)	92	(0.85570002, 0.13220000)
43	(0.67979997, 0.15480000)	93	(0.89389998, 0.26159999)
44	(0.51980001, 0.86220002)	94	(0.59350002, 0.23830000)
45	(0.15400000, 0.38180000)	95	(0.40650001, 0.21560000)
46	(0.43920001, 0.65270001)	96	(0.42739999, 0.59780002)
47	(0.44440001, 0.31650001)	97	(0.18359999, 0.90609998)
48	(0.16050000, 0.30190000)	98	(0.26620001, 0.83880001)
49	(0.58260000, 0.11800000)	99	(0.74710000, 0.71810001)
50	(0.12790000, 0.77509999)	100	(0.62010002, 0.50680000)

Optimal Steiner Points

1	(0.07195942, 0.83462816)	22	(0.01323256, 0.21001352)
2	(0.09190980, 0.78471529)	23	(0.02436871, 0.16545367)
3	(0.03500639, 0.71243823)	24	(0.14714077, 0.23809958)
4	(0.14074934, 0.91467291)	25	(0.16762093, 0.25440601)
5	(0.26285210, 0.87077653)	26	(0.90751314, 0.25495481)
6	(0.66712749, 0.97004169)	27	(0.88711274, 0.12836054)
7	(0.89723802, 0.82385200)	28	(0.92446512, 0.17798556)
8	(0.91382748, 0.84857035)	29	(0.68464303, 0.43338802)
9	(0.73279274, 0.69816387)	30	(0.67248875, 0.41661885)
10	(0.76153737, 0.73380679)	31	(0.59200341, 0.37119284)
11	(0.79179227, 0.72705036)	32	(0.60325956, 0.23622674)
12	(0.62404698, 0.71247178)	33	(0.62601304, 0.16622630)
13	(0.59236747, 0.67166191)	34	(0.70816332, 0.09139054)
14	(0.18186849, 0.41224524)	35	(0.54177243, 0.10532154)
15	(0.16728023, 0.45842862)	36	(0.41677481, 0.21751334)
16	(0.21668090, 0.51239693)	37	(0.44033733, 0.28423160)
17	(0.25650030, 0.50359112)	38	(0.43230900, 0.31273296)
18	(0.28881258, 0.53889102)	39	(0.26142073, 0.18335445)
19	(0.36799562, 0.52138025)	40	(0.20840517, 0.09734409)
20	(0.13530554, 0.45796323)	41	(0.22855642, 0.09738006)
21	(0.10673677, 0.47459298)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	35	60.3%
3	10	17.2%
4	11	19.0%
5	1	1.7%
6	0	0.0%
7	0	0.0%
8	1	1.7%

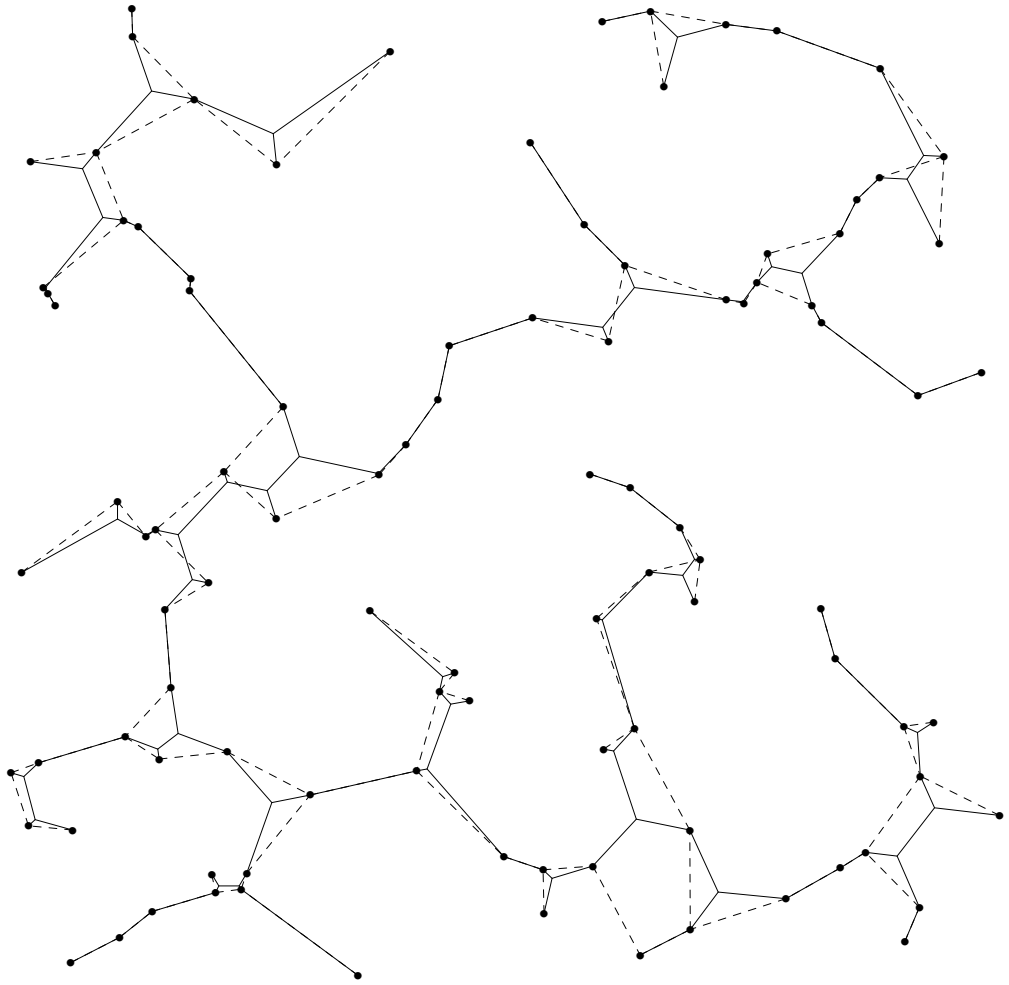


Figure B.25. Cockayne and Hewgill's Test Problem 25 Steiner minimal tree and minimum spanning tree.

B.26 Test Problem 26

Minimum Spanning Tree	6.4513
Steiner Minimal Tree	6.2851
Reduction	2.58%

Given Points

1	(0.03370000, 0.00280000)	51	(0.23549999, 0.25560001)
2	(0.06170000, 0.08680000)	52	(0.70340002, 0.81470001)
3	(0.06470000, 0.13590001)	53	(0.24730000, 0.55129999)
4	(0.03610000, 0.24349999)	54	(0.39240000, 0.86260003)
5	(0.01680000, 0.32720000)	55	(0.29830000, 0.63309997)
6	(0.03390000, 0.43689999)	56	(0.53839999, 0.13720000)
7	(0.12000000, 0.61430001)	57	(0.72289997, 0.48089999)
8	(0.21230000, 0.86949998)	58	(0.78070003, 0.54030001)
9	(0.19640000, 0.95150000)	59	(0.44589999, 0.53740001)
10	(0.18880001, 0.99550003)	60	(0.83480000, 0.68449998)
11	(0.32130000, 0.95279998)	61	(0.95090002, 0.44000000)
12	(0.37200001, 0.99280000)	62	(0.69830000, 0.83990002)
13	(0.42230001, 0.91289997)	63	(0.68409997, 0.41729999)
14	(0.58029997, 0.86049998)	64	(0.18140000, 0.47470000)
15	(0.68610001, 0.88110000)	65	(0.79850000, 0.41760001)
16	(0.84890002, 0.83069998)	66	(0.04670000, 0.32179999)
17	(0.98089999, 0.94679999)	67	(0.88650000, 0.77399999)
18	(0.90939999, 0.77939999)	68	(0.19960000, 0.38580000)
19	(0.90979999, 0.56290001)	69	(0.41490000, 0.14790000)
20	(0.96280003, 0.51779997)	70	(0.20670000, 0.53219998)
21	(0.99409997, 0.44659999)	71	(0.42379999, 0.59950000)
22	(0.99250001, 0.33770001)	72	(0.49980000, 0.56690001)
23	(0.92900002, 0.22679999)	73	(0.73549998, 0.10690000)
24	(0.95279998, 0.12580000)	74	(0.29969999, 0.41220000)
25	(0.98900002, 0.10700000)	75	(0.62220001, 0.69599998)
26	(0.95679998, 0.08220000)	76	(0.25070000, 0.76319999)
27	(0.91799998, 0.02750000)	77	(0.65399998, 0.24810000)
28	(0.87500000, 0.02950000)	78	(0.27000001, 0.17930000)
29	(0.74500000, 0.03930000)	79	(0.93210000, 0.50269997)
30	(0.73619998, 0.02950000)	80	(0.49300000, 0.31020001)
31	(0.60369998, 0.04130000)	81	(0.27410001, 0.23270001)
32	(0.59960002, 0.04560000)	82	(0.52219999, 0.44610000)
33	(0.53850001, 0.04250000)	83	(0.30620000, 0.28189999)
34	(0.44330001, 0.07720000)	84	(0.50720000, 0.76560003)
35	(0.35499999, 0.15500000)	85	(0.82480001, 0.08520000)
36	(0.32690001, 0.14970000)	86	(0.66740000, 0.10040000)
37	(0.16140001, 0.07990000)	87	(0.32409999, 0.50349998)
38	(0.07490000, 0.07590000)	88	(0.39179999, 0.40320000)
39	(0.64590001, 0.68129998)	89	(0.35249999, 0.47799999)
40	(0.91600001, 0.30960000)	90	(0.69489998, 0.73790002)
41	(0.80059999, 0.64859998)	91	(0.45249999, 0.42789999)
42	(0.56989998, 0.28680000)	92	(0.52209997, 0.35730001)
43	(0.48710001, 0.43189999)	93	(0.06690000, 0.36669999)
44	(0.56680000, 0.39160001)	94	(0.69199997, 0.85790002)
45	(0.82330000, 0.63129997)	95	(0.60900003, 0.22530000)
46	(0.71600002, 0.79799998)	96	(0.26120001, 0.26080000)
47	(0.49660000, 0.27039999)	97	(0.56919998, 0.12989999)
48	(0.83770001, 0.56430000)	98	(0.59399998, 0.82789999)
49	(0.59399998, 0.05710000)	99	(0.44150001, 0.81760001)
50	(0.73299998, 0.45150000)	100	(0.40160000, 0.42330000)

Optimal Steiner Points

1	(0.72761053, 0.45348820)	22	(0.27353796, 0.18129140)
2	(0.22531360, 0.92416716)	23	(0.45196652, 0.11223966)
3	(0.39794087, 0.86390465)	24	(0.56381893, 0.13833103)
4	(0.41195345, 0.91046357)	25	(0.60493529, 0.08849120)
5	(0.36291030, 0.96259785)	26	(0.59274685, 0.05338743)
6	(0.68254209, 0.86712414)	27	(0.61339867, 0.23917001)
7	(0.59792358, 0.84331197)	28	(0.49411857, 0.31018621)
8	(0.50731403, 0.77884257)	29	(0.50680226, 0.28757814)
9	(0.81357980, 0.64848512)	30	(0.39197758, 0.41789842)
10	(0.69479972, 0.71019530)	31	(0.35206005, 0.44159243)
11	(0.64590323, 0.68220025)	32	(0.46543807, 0.43924969)
12	(0.89351541, 0.78227955)	33	(0.54038048, 0.39092314)
13	(0.87804931, 0.82541561)	34	(0.51486635, 0.43261138)
14	(0.82649142, 0.57001573)	35	(0.45493606, 0.56025368)
15	(0.93936116, 0.51672447)	36	(0.27456123, 0.55738926)
16	(0.92254305, 0.30649760)	37	(0.19838072, 0.53027773)
17	(0.97780663, 0.34466693)	38	(0.16627660, 0.41949940)
18	(0.97084910, 0.43054113)	39	(0.07480076, 0.39553776)
19	(0.96739483, 0.10505921)	40	(0.03632101, 0.31584987)
20	(0.75974786, 0.07595895)	41	(0.06928910, 0.07551537)
21	(0.27146173, 0.25631157)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	28	48.3%
3	21	36.2%
4	7	12.1%
5	2	3.4%

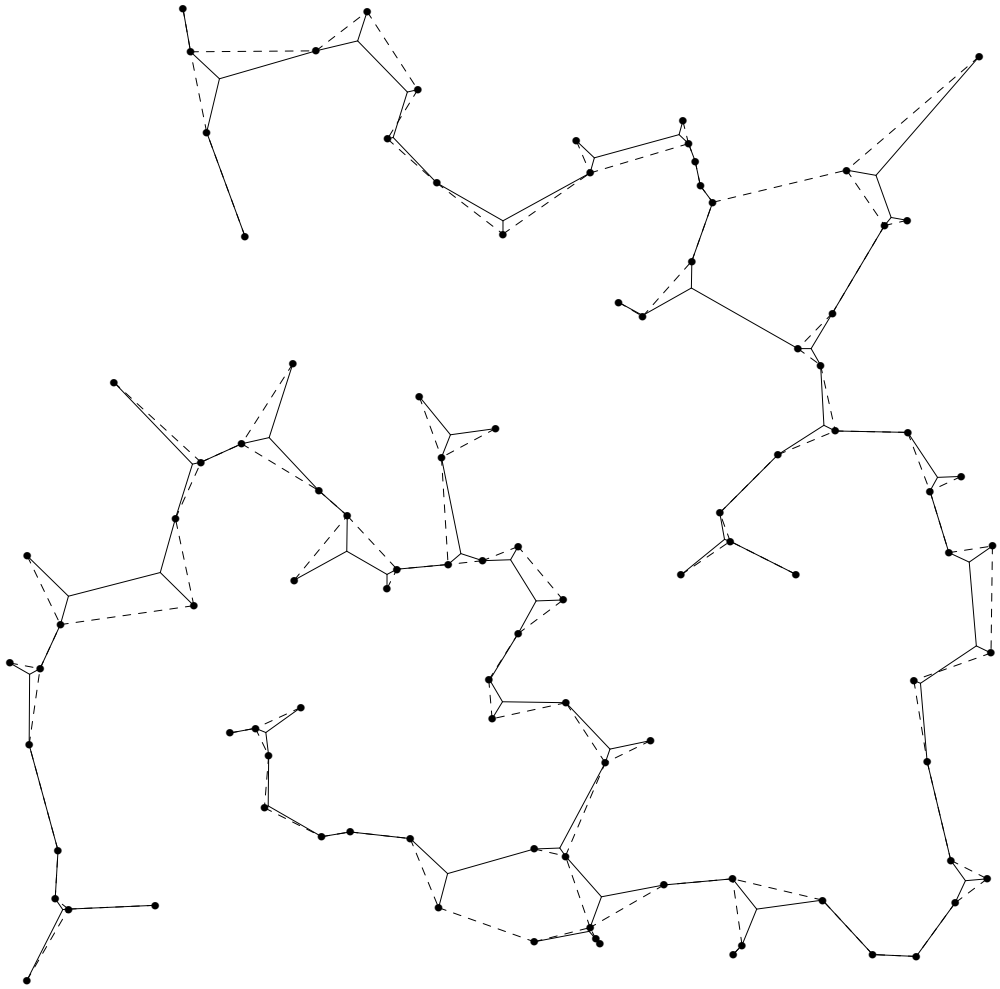


Figure B.26. Cockayne and Hewgill's Test Problem 26 Steiner minimal tree and minimum spanning tree.

B.27 Test Problem 27

Minimum Spanning Tree	6.8434
Steiner Minimal Tree	6.6462
Reduction	2.88%

Given Points

1	(0.56760001, 0.03110000)	51	(0.32020000, 0.44090000)
2	(0.40770000, 0.05380000)	52	(0.37439999, 0.40099999)
3	(0.26660001, 0.04150000)	53	(0.10070000, 0.27340001)
4	(0.17030001, 0.03720000)	54	(0.84050000, 0.52139997)
5	(0.15350001, 0.04250000)	55	(0.14730000, 0.23670000)
6	(0.02820000, 0.03320000)	56	(0.90020001, 0.33379999)
7	(0.00000000, 0.21410000)	57	(0.67040002, 0.11090000)
8	(0.04810000, 0.34509999)	58	(0.47749999, 0.53680003)
9	(0.04600000, 0.49720001)	59	(0.88209999, 0.21040000)
10	(0.02710000, 0.55470002)	60	(0.18330000, 0.10130000)
11	(0.02780000, 0.59640002)	61	(0.69220001, 0.77350003)
12	(0.06860000, 0.61610001)	62	(0.41490000, 0.87120003)
13	(0.08990000, 0.66900003)	63	(0.65859997, 0.62790000)
14	(0.08160000, 0.69520003)	64	(0.85380000, 0.52370000)
15	(0.01560000, 0.92580003)	65	(0.33590001, 0.81250000)
16	(0.29980001, 0.99849999)	66	(0.31540000, 0.66850001)
17	(0.41710001, 0.97049999)	67	(0.32600001, 0.72430003)
18	(0.85070002, 0.91339999)	68	(0.51169997, 0.41170001)
19	(0.98199999, 0.96689999)	69	(0.36460000, 0.13869999)
20	(0.92430001, 0.89270002)	70	(0.55800003, 0.44670001)
21	(0.94040000, 0.78839999)	71	(0.41740000, 0.80409998)
22	(0.95359999, 0.75760001)	72	(0.31389999, 0.25470001)
23	(0.91439998, 0.66630000)	73	(0.62059999, 0.11210000)
24	(0.92720002, 0.63029999)	74	(0.07450000, 0.17739999)
25	(0.94430000, 0.45350000)	75	(0.12300000, 0.77929997)
26	(0.98369998, 0.32519999)	76	(0.16410001, 0.05890000)
27	(0.97350001, 0.18910000)	77	(0.91930002, 0.40750000)
28	(0.94040000, 0.09090000)	78	(0.17850000, 0.09380000)
29	(0.92240000, 0.08080000)	79	(0.73040003, 0.17340000)
30	(0.84969997, 0.08150000)	80	(0.25029999, 0.53850001)
31	(0.75379997, 0.03360000)	81	(0.70819998, 0.56559998)
32	(0.56260002, 0.45300001)	82	(0.79350001, 0.36939999)
33	(0.91469997, 0.21799999)	83	(0.71230000, 0.75250000)
34	(0.88110000, 0.16540000)	84	(0.93559998, 0.09470000)
35	(0.35159999, 0.81519997)	85	(0.43000001, 0.70959997)
36	(0.58179998, 0.13190000)	86	(0.32949999, 0.51249999)
37	(0.40430000, 0.31529999)	87	(0.28900000, 0.17110001)
38	(0.59539998, 0.31240001)	88	(0.65120000, 0.80629998)
39	(0.51050001, 0.15660000)	89	(0.61350000, 0.59350002)
40	(0.25520000, 0.48609999)	90	(0.21720000, 0.20080000)
41	(0.23320000, 0.13440000)	91	(0.62819999, 0.52370000)
42	(0.44170001, 0.40979999)	92	(0.10350000, 0.14780000)
43	(0.38299999, 0.52399999)	93	(0.79460001, 0.59810001)
44	(0.23310000, 0.42800000)	94	(0.66970003, 0.80640000)
45	(0.15340000, 0.31680000)	95	(0.70480001, 0.44800001)
46	(0.85879999, 0.88720000)	96	(0.79820001, 0.20980000)
47	(0.72149998, 0.09910000)	97	(0.51490003, 0.57950002)
48	(0.93660003, 0.34549999)	98	(0.22540000, 0.88720000)
49	(0.26789999, 0.90740001)	99	(0.38190001, 0.82050002)
50	(0.40210000, 0.77480000)	100	(0.71310002, 0.43650001)

Optimal Steiner Points

1	(0.02900284, 0.59562898)	20	(0.93567187, 0.09331275)
2	(0.12947047, 0.84392208)	21	(0.92454535, 0.35104257)
3	(0.32796377, 0.72246194)	22	(0.89011675, 0.32646346)
4	(0.39462808, 0.74270791)	23	(0.90094590, 0.22411239)
5	(0.39674693, 0.82295287)	24	(0.87014025, 0.19641618)
6	(0.41284075, 0.80334675)	25	(0.62839180, 0.49150938)
7	(0.39156169, 0.94185823)	26	(0.65333343, 0.56618309)
8	(0.31313396, 0.95813078)	27	(0.63809395, 0.59323865)
9	(0.30699310, 0.17939848)	28	(0.70893091, 0.75203073)
10	(0.24542028, 0.16486454)	29	(0.84738117, 0.52649134)
11	(0.22023261, 0.10521173)	30	(0.84155113, 0.57768244)
12	(0.16131663, 0.04512681)	31	(0.91804653, 0.63428050)
13	(0.10965808, 0.24502891)	32	(0.53787422, 0.41173458)
14	(0.06763756, 0.19913402)	33	(0.40246186, 0.38576010)
15	(0.10484865, 0.29852694)	34	(0.30388790, 0.47188836)
16	(0.91066372, 0.87728858)	35	(0.26181546, 0.47353253)
17	(0.85928243, 0.88774520)	36	(0.70779985, 0.11640321)
18	(0.88428390, 0.10392905)	37	(0.59943014, 0.10919104)
19	(0.92220432, 0.08457848)		

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	37	59.7%
3	15	24.2%
4	8	12.9%
5	2	3.2%

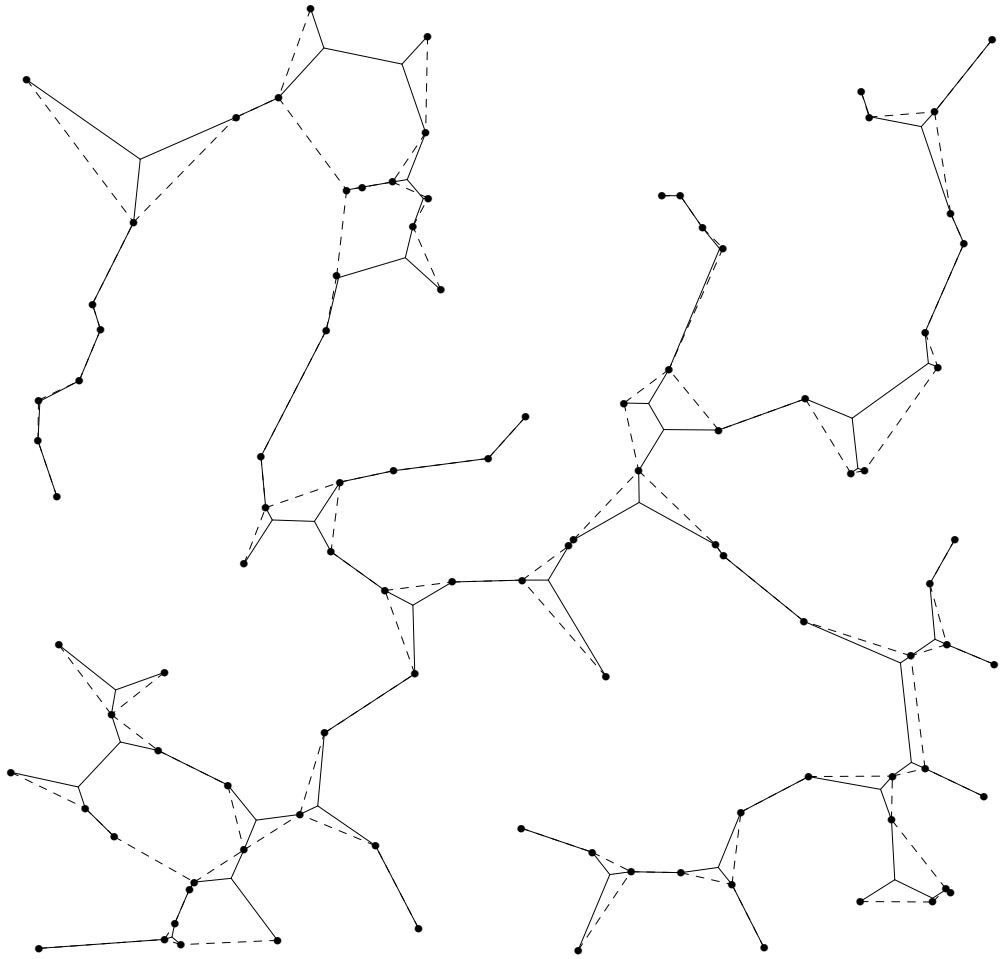


Figure B.27. Cockayne and Hewgill's Test Problem 27 Steiner minimal tree and minimum spanning tree.

B.28 Test Problem 28

Minimum Spanning Tree	6.6698
Steiner Minimal Tree	6.4592
Reduction	3.16%

Given Points

1	(0.46020001, 0.01540000)	51	(0.07970000, 0.60780001)
2	(0.32339999, 0.04880000)	52	(0.71280003, 0.76429999)
3	(0.20670000, 0.04040000)	53	(0.37830001, 0.44270000)
4	(0.17030001, 0.01830000)	54	(0.10100000, 0.28839999)
5	(0.15290000, 0.04910000)	55	(0.53469998, 0.04940000)
6	(0.07320000, 0.19400001)	56	(0.33109999, 0.53710002)
7	(0.00000000, 0.21420000)	57	(0.32720000, 0.76959997)
8	(0.06990000, 0.29789999)	58	(0.45750001, 0.61960000)
9	(0.06130000, 0.46219999)	59	(0.58160001, 0.34509999)
10	(0.00560000, 0.63639998)	60	(0.97259998, 0.67309999)
11	(0.00970000, 0.66159999)	61	(0.29429999, 0.45249999)
12	(0.04680000, 0.89340001)	62	(0.66170001, 0.42390001)
13	(0.22690000, 0.97390002)	63	(0.47040001, 0.42770001)
14	(0.22880000, 0.96719998)	64	(0.82499999, 0.35659999)
15	(0.26620001, 0.94379997)	65	(0.61110002, 0.39160001)
16	(0.36939999, 0.97380000)	66	(0.57910001, 0.11810000)
17	(0.43840000, 0.91320002)	67	(0.35609999, 0.44909999)
18	(0.50900000, 0.88239998)	68	(0.69730002, 0.90030003)
19	(0.56900001, 0.92519999)	69	(0.16320001, 0.24220000)
20	(0.67430001, 0.95850003)	70	(0.25470001, 0.36390001)
21	(0.68739998, 0.95490003)	71	(0.88810003, 0.11380000)
22	(0.73479998, 0.92430001)	72	(0.22730000, 0.45690000)
23	(0.77319998, 0.92940003)	73	(0.70090002, 0.70730001)
24	(0.89429998, 0.91710001)	74	(0.35120001, 0.16940001)
25	(0.94389999, 0.92019999)	75	(0.24670000, 0.56230003)
26	(0.96249998, 0.92079997)	76	(0.27489999, 0.83840001)
27	(0.98439997, 0.67449999)	77	(0.67530000, 0.77410001)
28	(0.99690002, 0.65429997)	78	(0.31619999, 0.47799999)
29	(0.97280002, 0.61040002)	79	(0.82910001, 0.11830000)
30	(0.96429998, 0.39770001)	80	(0.85350001, 0.12080000)
31	(0.93889999, 0.22090000)	81	(0.22690000, 0.86710000)
32	(0.96410000, 0.08590000)	82	(0.39879999, 0.63370001)
33	(0.82380003, 0.03740000)	83	(0.95359999, 0.65390003)
34	(0.66350001, 0.08120000)	84	(0.57709998, 0.65410000)
35	(0.64120001, 0.06010000)	85	(0.84320003, 0.27689999)
36	(0.53090000, 0.01690000)	86	(0.49610001, 0.75370002)
37	(0.74449998, 0.54250002)	87	(0.70150000, 0.91670001)
38	(0.33989999, 0.21470000)	88	(0.71340001, 0.64850003)
39	(0.32859999, 0.36809999)	89	(0.34889999, 0.36120000)
40	(0.42150000, 0.41350001)	90	(0.67970002, 0.94739997)
41	(0.45500001, 0.37509999)	91	(0.69389999, 0.22420000)
42	(0.78570002, 0.45570001)	92	(0.87599999, 0.36379999)
43	(0.26179999, 0.21390000)	93	(0.83999997, 0.52230000)
44	(0.45420000, 0.26320001)	94	(0.62620002, 0.32900000)
45	(0.64780003, 0.13320000)	95	(0.41060001, 0.56370002)
46	(0.23480000, 0.80390000)	96	(0.70690000, 0.54689997)
47	(0.61500001, 0.66200000)	97	(0.34770000, 0.42429999)
48	(0.20400000, 0.44610000)	98	(0.10200000, 0.55040002)
49	(0.38690001, 0.17659999)	99	(0.72310001, 0.63639998)
50	(0.62059999, 0.09310000)	100	(0.57020003, 0.60759997)

Optimal Steiner Points

1	(0.98404181, 0.67378414)	20	(0.68079853, 0.95271415)
2	(0.95905811, 0.61590350)	21	(0.70246947, 0.91921687)
3	(0.24211943, 0.94390219)	22	(0.68419850, 0.77804023)
4	(0.21226580, 0.89269745)	23	(0.70671958, 0.76160765)
5	(0.24928233, 0.83587366)	24	(0.68985987, 0.67598355)
6	(0.01137536, 0.64037108)	25	(0.56425065, 0.64354640)
7	(0.33742830, 0.37115046)	26	(0.51101917, 0.66355735)
8	(0.11401966, 0.25141132)	27	(0.72231287, 0.55559957)
9	(0.07075440, 0.20094790)	28	(0.79296184, 0.50384462)
10	(0.17307566, 0.03020177)	29	(0.84377337, 0.34555107)
11	(0.31321326, 0.06062229)	30	(0.91335475, 0.12441655)
12	(0.35623822, 0.17753986)	31	(0.92479283, 0.21496958)
13	(0.33795989, 0.21156555)	32	(0.44752282, 0.40586987)
14	(0.83510411, 0.11353344)	33	(0.48517638, 0.33422327)
15	(0.34815425, 0.44474486)	34	(0.60198128, 0.35141495)
16	(0.31589642, 0.46433699)	35	(0.63389945, 0.09539954)
17	(0.31436288, 0.52657402)	36	(0.64797193, 0.07851509)
18	(0.42272532, 0.61005896)	37	(0.58401006, 0.10174334)
19	(0.10636288, 0.49299291)	38	(0.52479249, 0.02467035)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	31	50.8%
3	22	36.1%
4	8	13.1%

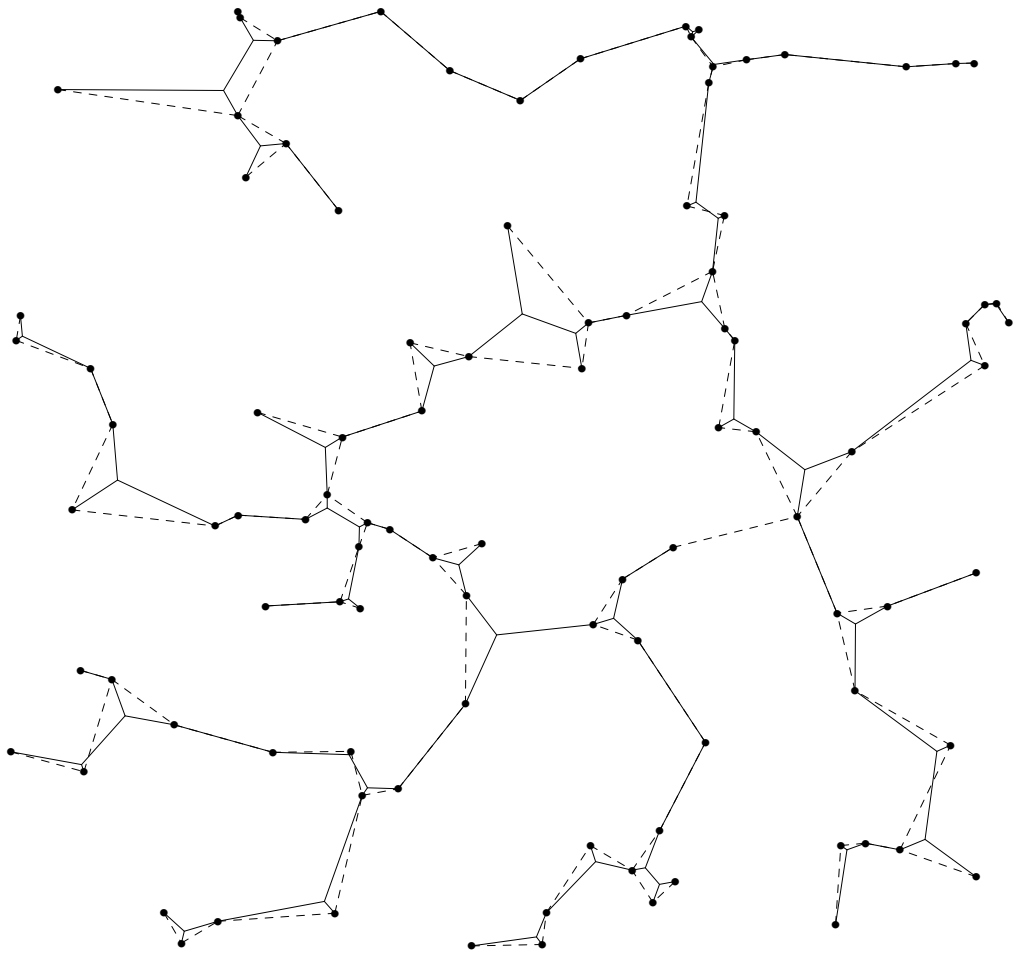


Figure B.28. Cockayne and Hewgill's Test Problem 28 Steiner minimal tree and minimum spanning tree.

B.29 Test Problem 29

Minimum Spanning Tree	7.1392
Steiner Minimal Tree	6.8628
Reduction	3.87%

Given Points

1	(0.12520000, 0.00070000)	51	(0.16249999, 0.42940000)
2	(0.01440000, 0.06320000)	52	(0.66890001, 0.43309999)
3	(0.00290000, 0.18130000)	53	(0.64359999, 0.33610001)
4	(0.00000000, 0.21430001)	54	(0.27500001, 0.63129997)
5	(0.01330000, 0.22080000)	55	(0.90579998, 0.62739998)
6	(0.01870000, 0.24089999)	56	(0.51590002, 0.89099997)
7	(0.05800000, 0.36059999)	57	(0.28500000, 0.60720003)
8	(0.02230000, 0.42230001)	58	(0.44760001, 0.40750000)
9	(0.04120000, 0.78920001)	59	(0.55460000, 0.77850002)
10	(0.06730000, 0.84909999)	60	(0.83069998, 0.21510001)
11	(0.01660000, 0.92290002)	61	(0.62519997, 0.65149999)
12	(0.18460000, 0.93550003)	62	(0.10400000, 0.73280001)
13	(0.28240001, 0.89639997)	63	(0.12800001, 0.52670002)
14	(0.45339999, 0.97890002)	64	(0.76109999, 0.41450000)
15	(0.62339997, 0.94330001)	65	(0.80049998, 0.22100000)
16	(0.65770000, 0.98030001)	66	(0.88810003, 0.36700001)
17	(0.75669998, 0.93849999)	67	(0.64389998, 0.42559999)
18	(0.83450001, 0.99010003)	68	(0.96730000, 0.66560000)
19	(0.98409998, 0.83490002)	69	(0.48379999, 0.31299999)
20	(0.99809998, 0.68250000)	70	(0.72799999, 0.11590000)
21	(0.96869999, 0.59230000)	71	(0.58380002, 0.81510001)
22	(0.94129997, 0.32760000)	72	(0.52499998, 0.52440000)
23	(0.99280000, 0.21619999)	73	(0.30199999, 0.55190003)
24	(0.95959997, 0.07940000)	74	(0.90710002, 0.74820000)
25	(0.80419999, 0.07820000)	75	(0.22220001, 0.16869999)
26	(0.75580001, 0.00220000)	76	(0.65840000, 0.09260000)
27	(0.64349997, 0.05200000)	77	(0.57190001, 0.77300000)
28	(0.46430001, 0.08040000)	78	(0.83480000, 0.50629997)
29	(0.27480000, 0.00800000)	79	(0.65869999, 0.65230000)
30	(0.70760000, 0.46520001)	80	(0.44499999, 0.50569999)
31	(0.36700001, 0.77969998)	81	(0.61379999, 0.12270000)
32	(0.22700000, 0.46239999)	82	(0.27390000, 0.17160000)
33	(0.10080000, 0.24439999)	83	(0.34189999, 0.44679999)
34	(0.22950000, 0.41659999)	84	(0.62400001, 0.30870000)
35	(0.42879999, 0.67490000)	85	(0.32269999, 0.11870000)
36	(0.74260002, 0.41670001)	86	(0.81919998, 0.47040001)
37	(0.60210001, 0.37059999)	87	(0.76650000, 0.66570002)
38	(0.40480000, 0.11400000)	88	(0.68870002, 0.74339998)
39	(0.20350000, 0.66490000)	89	(0.44670001, 0.48040000)
40	(0.35360000, 0.68550003)	90	(0.82099998, 0.16329999)
41	(0.32690001, 0.39530000)	91	(0.46030000, 0.21070001)
42	(0.91930002, 0.25680000)	92	(0.41650000, 0.24250001)
43	(0.47650000, 0.68599999)	93	(0.38620001, 0.10960000)
44	(0.86240000, 0.31709999)	94	(0.56660002, 0.29879999)
45	(0.79369998, 0.50830001)	95	(0.05570000, 0.12690000)
46	(0.64270002, 0.59670001)	96	(0.44549999, 0.35130000)
47	(0.25549999, 0.76709998)	97	(0.37709999, 0.18310000)
48	(0.95310003, 0.69679999)	98	(0.85850000, 0.60579997)
49	(0.67629999, 0.77450001)	99	(0.67070001, 0.35460001)
50	(0.21150000, 0.58410001)	100	(0.80610001, 0.36620000)

Optimal Steiner Points

1	(0.65104330, 0.09009562)	23	(0.31723744, 0.11784685)
2	(0.78721708, 0.11227165)	24	(0.36419883, 0.13072658)
3	(0.82083464, 0.20935693)	25	(0.42079303, 0.22314978)
4	(0.87810528, 0.26619893)	26	(0.46398282, 0.31421509)
5	(0.97835988, 0.21040943)	27	(0.04958661, 0.79113722)
6	(0.86147225, 0.33750233)	28	(0.06038254, 0.26639175)
7	(0.88865089, 0.35488632)	29	(0.06248225, 0.39021587)
8	(0.92733103, 0.74717259)	30	(0.16391233, 0.52199656)
9	(0.97098255, 0.67682016)	31	(0.18834332, 0.46312642)
10	(0.94684547, 0.62830216)	32	(0.22398014, 0.45845902)
11	(0.64192003, 0.64222491)	33	(0.23718791, 0.42663297)
12	(0.65894455, 0.97346455)	34	(0.31830588, 0.41600886)
13	(0.56408489, 0.87982345)	35	(0.42613113, 0.44576424)
14	(0.57141495, 0.77459419)	36	(0.45094270, 0.50034779)
15	(0.59005785, 0.79453254)	37	(0.23081714, 0.62688792)
16	(0.69970238, 0.68831575)	38	(0.28183588, 0.62859517)
17	(0.84828967, 0.60168499)	39	(0.37010112, 0.69845492)
18	(0.81808001, 0.49563107)	40	(0.35934633, 0.77369118)
19	(0.78210711, 0.41338220)	41	(0.29618114, 0.79903841)
20	(0.70480204, 0.44705135)	42	(0.28227732, 0.89630365)
21	(0.64847296, 0.35027033)	43	(0.00163565, 0.21343954)
22	(0.62662953, 0.37537819)	44	(0.04440776, 0.07304198)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	21	38.2%
3	29	52.7%
4	3	5.5%
5	0	0.0%
6	1	1.8%
7	1	1.8%

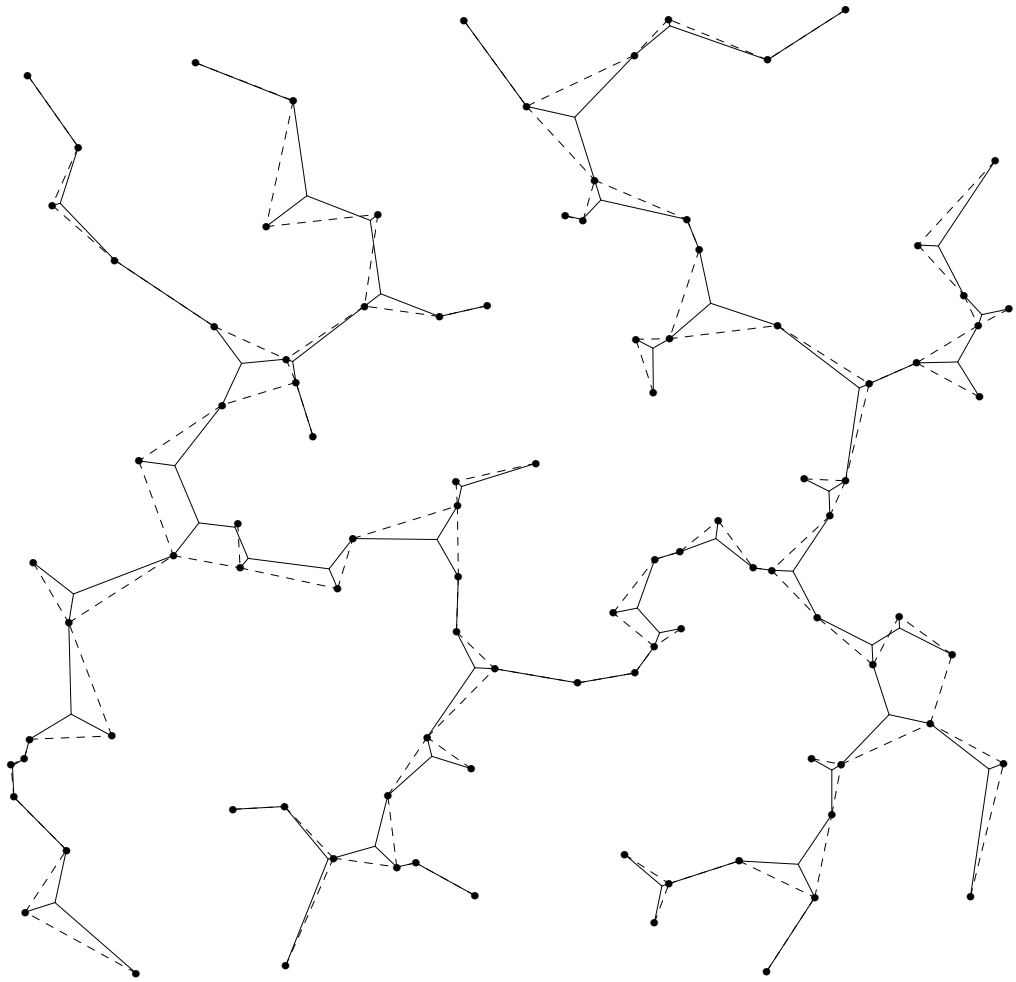


Figure B.29. Cockayne and Hewgill's Test Problem 29 Steiner minimal tree and minimum spanning tree.

B.30 Test Problem 30

Minimum Spanning Tree	6.4715
Steiner Minimal Tree	6.2736
Reduction	3.06%

Given Points

1	(0.31819999, 0.01050000)	51	(0.53490001, 0.50880003)
2	(0.15989999, 0.11050000)	52	(0.86659998, 0.14229999)
3	(0.09880000, 0.19110000)	53	(0.83020002, 0.65300000)
4	(0.07490000, 0.19570000)	54	(0.44520000, 0.62140000)
5	(0.00000000, 0.21439999)	55	(0.72530001, 0.58149999)
6	(0.05770000, 0.32370001)	56	(0.49649999, 0.44020000)
7	(0.01840000, 0.53530002)	57	(0.80430001, 0.27140000)
8	(0.02170000, 0.79070002)	58	(0.39870000, 0.18610001)
9	(0.12080000, 0.96079999)	59	(0.61820000, 0.55220002)
10	(0.22130001, 0.90549999)	60	(0.72920001, 0.93169999)
11	(0.32969999, 0.88120002)	61	(0.57760000, 0.29420000)
12	(0.45019999, 0.82810003)	62	(0.50400001, 0.52179998)
13	(0.61839998, 0.87199998)	63	(0.28900000, 0.55440003)
14	(0.69239998, 0.96910000)	64	(0.12090000, 0.79780000)
15	(0.78590000, 0.97100002)	65	(0.62159997, 0.54879999)
16	(0.84939998, 0.94400001)	66	(0.71380001, 0.27280000)
17	(0.84069997, 0.87070000)	67	(0.20039999, 0.64829999)
18	(0.87519997, 0.82550001)	68	(0.73470002, 0.80190003)
19	(0.96990001, 0.73500001)	69	(0.83980000, 0.48519999)
20	(0.95880002, 0.71899998)	70	(0.59109998, 0.22059999)
21	(0.96340001, 0.63239998)	71	(0.18619999, 0.72600001)
22	(0.97839999, 0.60780001)	72	(0.63040000, 0.48590001)
23	(0.92449999, 0.54900002)	73	(0.69400001, 0.40300000)
24	(0.89330000, 0.49050000)	74	(0.55989999, 0.77740002)
25	(0.91560000, 0.28709999)	75	(0.07420000, 0.54369998)
26	(0.94590002, 0.08540000)	76	(0.13660000, 0.54540002)
27	(0.95469999, 0.03010000)	77	(0.57940000, 0.38450000)
28	(0.82190001, 0.06730000)	78	(0.78289998, 0.36950001)
29	(0.80489999, 0.06150000)	79	(0.81639999, 0.64709997)
30	(0.77120000, 0.04890000)	80	(0.48249999, 0.56269997)
31	(0.54860002, 0.01310000)	81	(0.51779997, 0.18060000)
32	(0.19499999, 0.74100000)	82	(0.82819998, 0.86350000)
33	(0.37959999, 0.53939998)	83	(0.34390000, 0.78829998)
34	(0.21349999, 0.75919998)	84	(0.62099999, 0.39919999)
35	(0.58670002, 0.44060001)	85	(0.68750000, 0.58899999)
36	(0.88090003, 0.58700001)	86	(0.15099999, 0.37639999)
37	(0.45550001, 0.10060000)	87	(0.20230000, 0.34310001)
38	(0.38330001, 0.56019998)	88	(0.62930000, 0.50639999)
39	(0.51529998, 0.32589999)	89	(0.48750001, 0.75559998)
40	(0.60900003, 0.55849999)	90	(0.53839999, 0.54759997)
41	(0.82880002, 0.87419999)	91	(0.89399999, 0.65530002)
42	(0.76480001, 0.60280001)	92	(0.11400000, 0.48830000)
43	(0.13910000, 0.76179999)	93	(0.86669999, 0.17500000)
44	(0.68570000, 0.15230000)	94	(0.86489999, 0.78630000)
45	(0.81320000, 0.09760000)	95	(0.80559999, 0.09440000)
46	(0.65789998, 0.15889999)	96	(0.84500003, 0.16740000)
47	(0.58929998, 0.57520002)	97	(0.35030001, 0.38330001)
48	(0.43720001, 0.32480001)	98	(0.43590000, 0.52920002)
49	(0.12780000, 0.46079999)	99	(0.65319997, 0.51080000)
50	(0.27329999, 0.76230001)	100	(0.77640003, 0.28029999)

Optimal Steiner Points

1	(0.11892650, 0.79408896)	20	(0.36400422, 0.82558882)
2	(0.03859857, 0.22751573)	21	(0.49359840, 0.76970571)
3	(0.13686892, 0.38251278)	22	(0.83149874, 0.87012976)
4	(0.11046983, 0.52695489)	23	(0.83377177, 0.93656510)
5	(0.89133435, 0.49296498)	24	(0.78578949, 0.96961391)
6	(0.91262746, 0.54721111)	25	(0.73011088, 0.94312578)
7	(0.87234563, 0.63736016)	26	(0.86678308, 0.12957281)
8	(0.93807560, 0.66286331)	27	(0.80866748, 0.09343681)
9	(0.95966136, 0.73010248)	28	(0.81393683, 0.06980053)
10	(0.87243849, 0.78990608)	29	(0.85833555, 0.16504171)
11	(0.63999796, 0.51550269)	30	(0.85943818, 0.24660890)
12	(0.63474971, 0.54411596)	31	(0.76056290, 0.35849342)
13	(0.58906782, 0.57206112)	32	(0.75629812, 0.29374093)
14	(0.37526453, 0.54830533)	33	(0.59829891, 0.40456209)
15	(0.47878200, 0.54383886)	34	(0.58753967, 0.44040164)
16	(0.52487618, 0.52243602)	35	(0.55333841, 0.32909882)
17	(0.53334218, 0.50875252)	36	(0.58983779, 0.21074086)
18	(0.20520735, 0.59345633)	37	(0.45386511, 0.07685030)
19	(0.18605141, 0.73564816)	38	(0.45745361, 0.14893682)

Number of Given Points in full Steiner Tree	Number of full Steiner Trees	Percentage
2	32	52.5%
3	21	34.4%
4	7	11.5%
5	1	1.6%

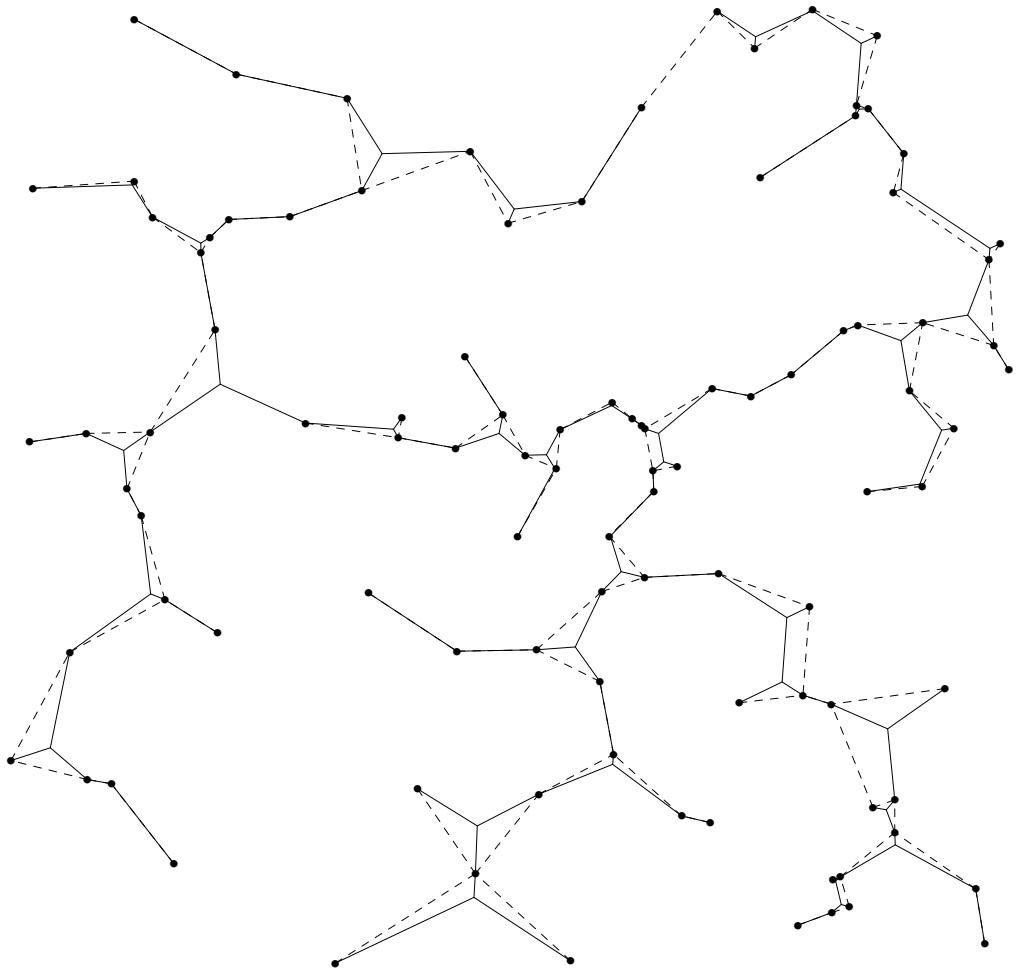


Figure B.30. Cockayne and Hewgill's Test Problem 30 Steiner minimal tree and minimum spanning tree.

Appendix C

An Example of Beasley and Goffinet's Heuristic

In Section 4.3 the heuristic by Beasley and Goffinet [2] is described. An example of using the heuristic is shown in this appendix. The same set of points is used as in the worked example in Section 4.2 which describes the heuristic by Smith *et al.* [37].

The heuristic is demonstrated by the liberal use of pictures showing the current solution, or the details of changes to a solution in the three different stages of the heuristic, expansion, reduction and re-expansion. Not all rules of the heuristic are used. In no situations are there any S-points with a degree of four or five. However this example does exhibit cycling. Because of this the example is not worked through to its completion, but stops when the cycling is first detected. One part of the reduction stage involves repeatedly moving S-points with degree three. This is necessary when S-points are dependent on other S-points, that is there is a full Steiner tree of four or more given points. The repetition is circumvented by moving the S-points to their optimal locations simultaneously, assuming the FST exists. The pleasing aspect of the example is that the heuristic finds the optimal solution.

C.1 First Iteration

The first Delaunay triangulation, added S-points and the MST spanning the given points V and the S-points S are shown in Figure C.1.[1] Many of the S-points are of degree two and in some instances of degree one in the MST.

Figure C.2 shows the MST of $V \cup S$ where any points in S with degree two or less in Figure C.1 have been removed. All S-points are in their optimal positions with respect to the points to which they are directly connected in the MST. This tree is very similar to the optimal solution shown in Figure 4.8.

The tree in Figure C.2 has two pairs of edges that make an angle of less than 120° with each other at a common point. This occurs at points 2 and 6. Two S-points are added to remove these occurrences. First a S-point is inserted into

[1] In all figures in this appendix S-points are shown as crosses.

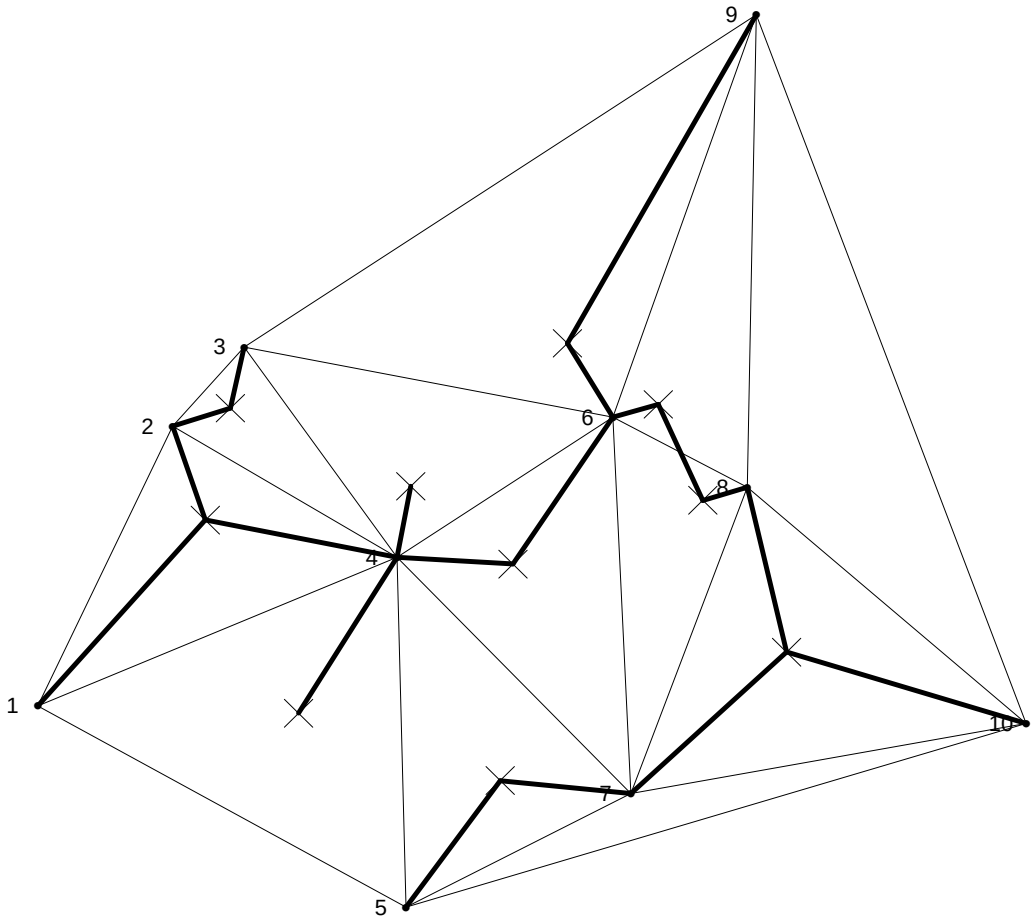


Figure C.1. The first expansion Delaunay triangulation and the S-points (crosses) of triangles. The bold line is the MST of the given points and the S-points.

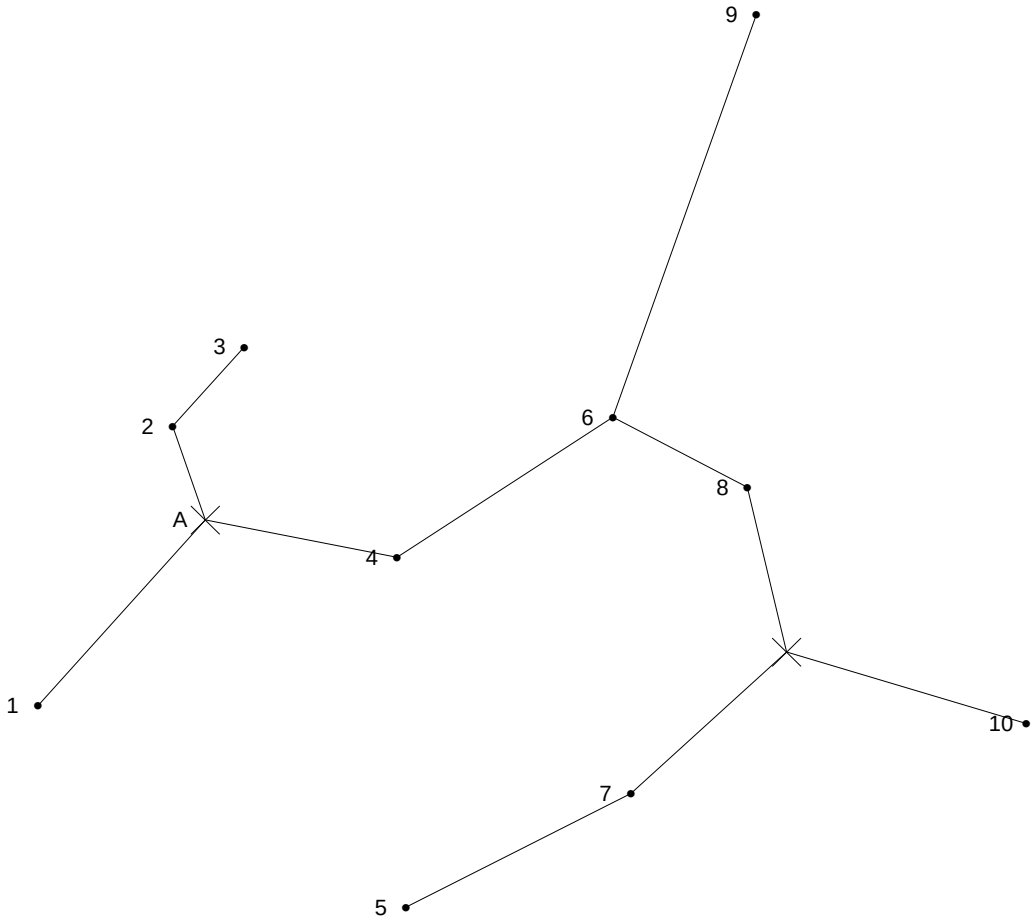


Figure C.2. Removing all degree one and two S-points in Figure C.1 leaves only two S-points. Both are in their optimal location.

the triangle formed by points 2, 3 and S-point A (shown in Figure C.2). And second, a S-point is added to the triangle (6,8,9). The first new S-point and A are now not in their optimal locations. The points are moved to the optimal locations given by the four point FST of points 1, 2, 3 and 4. Figure C.3 shows the solution at the end of the first iteration. There are four S-points, each in their optimal location and no edges of the MST make angles of less than 120°. This is in fact the optimal solution to the problem. Of course this is not actually known and the heuristic continues.

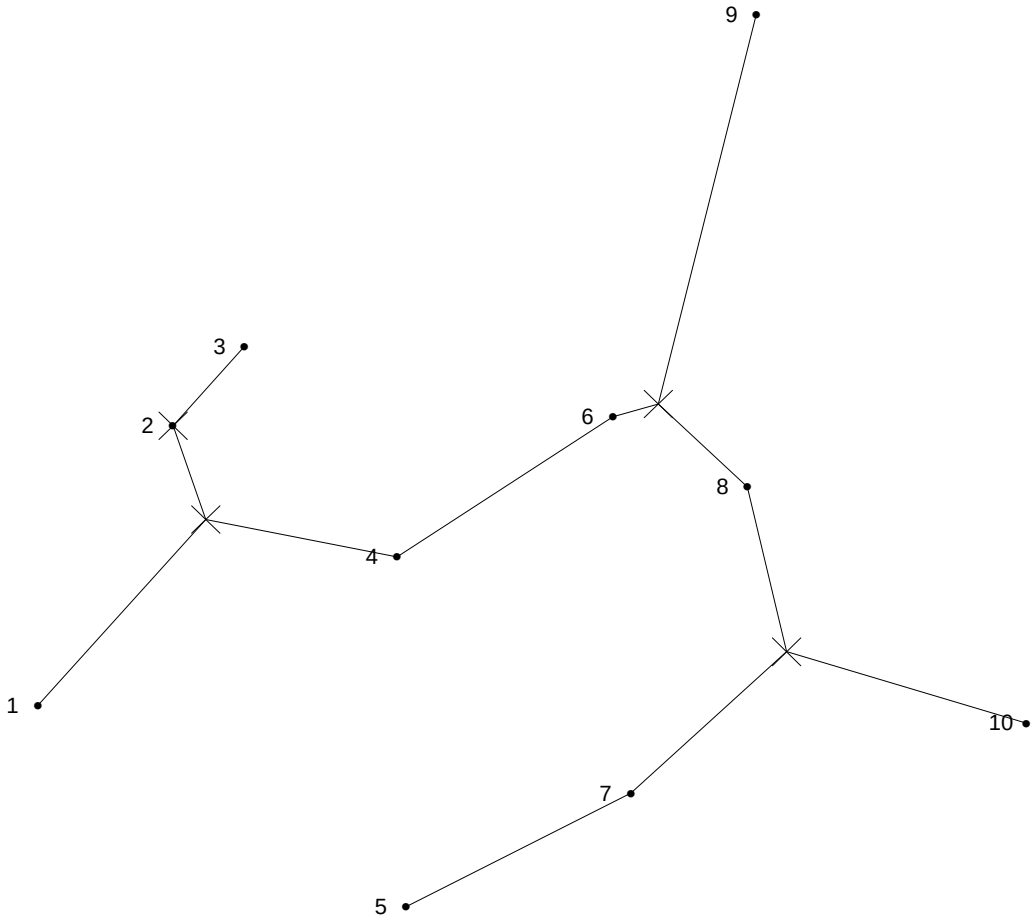


Figure C.3. The solution after adding further S-points to eliminate angles of less than 120° and moving S-points to their optimal locations.

C.2 Second Iteration

The length of the solution at the end of the first iteration is different to that at the beginning of the iteration therefore only one expansion is performed in the second iteration. Figure C.4 shows the solution at the end of the second iteration. It was only necessary to remove S-point of degree one and two and again add a S-points to the triangle (6,8,9) to remove an angle of less than 120° (as in the previous iteration). The solution is identical to that at the beginning of the iteration.

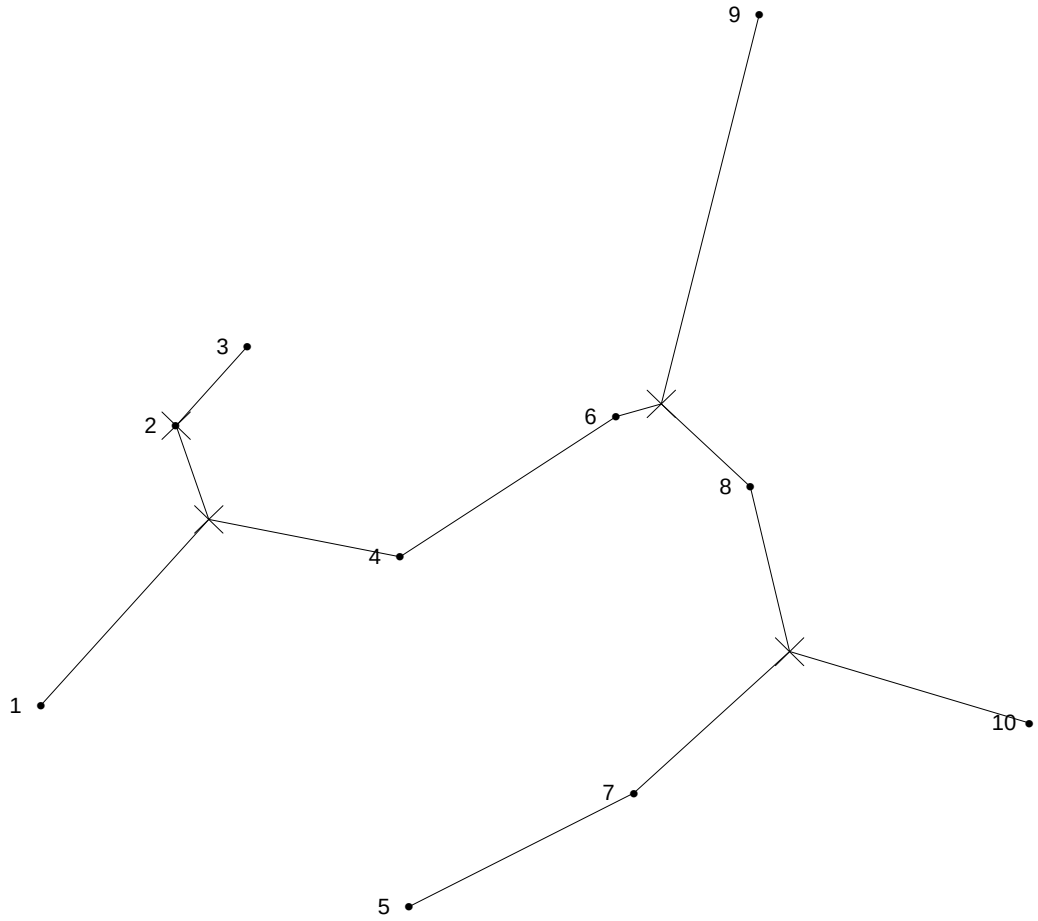


Figure C.4. The solution at the end of the second iteration is identical to that at the beginning of the iteration.

C.3 Third Iteration

The number of expansions increases to two because the previous iteration gave no change in solution. Figure C.5 shows the some what chaotic state of the solution after perform two Delaunay triangulations and adding S-points to triangles where possible.

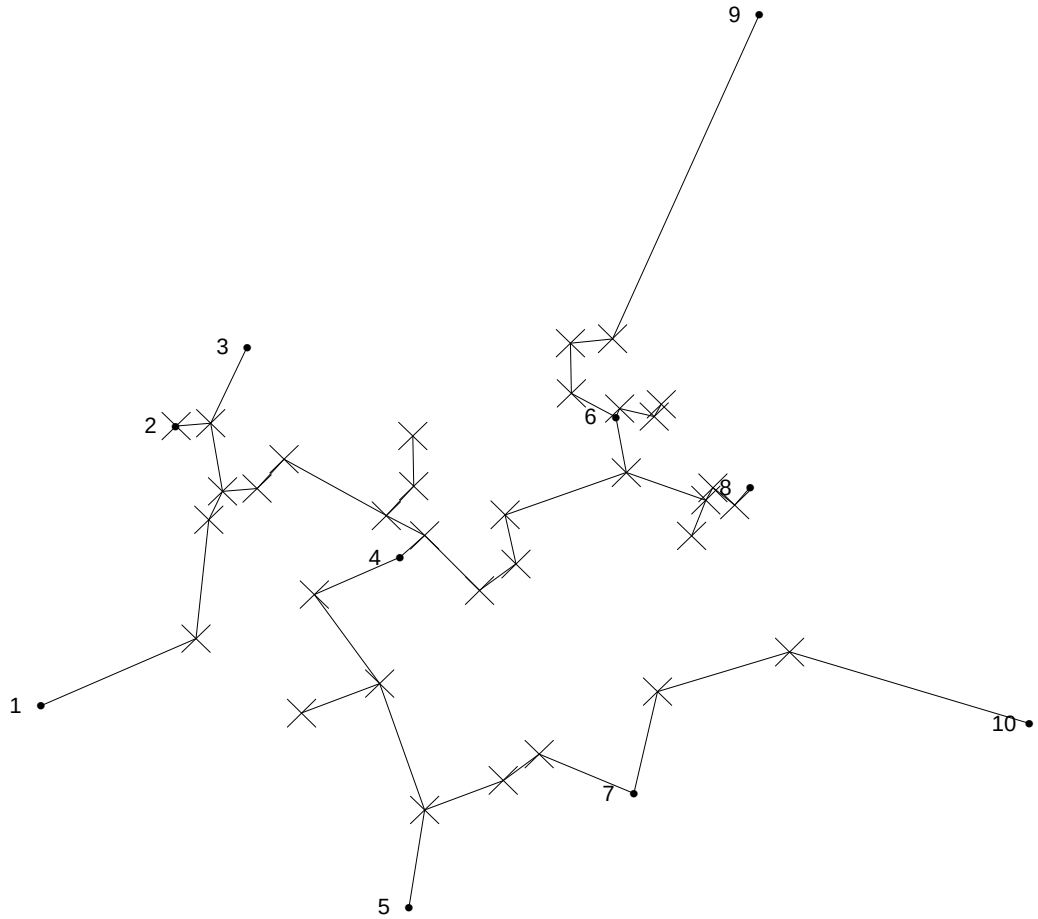


Figure C.5. The solution immediately after the two Delaunay triangulations and addition of many S-points. The vast majority of the S-points are of degree one and two.

Figure C.6 shows the solution after three rounds of removing degree one and two S-points. Four points survive this, but clearly none are in their optimal positions.

Figure C.7 shows the solution after the four S-points are moved to their optimal locations. Unfortunately moving the S-points creates an angle of less than 120°. An S-point is added to the triangle made by point 4 and S-points A

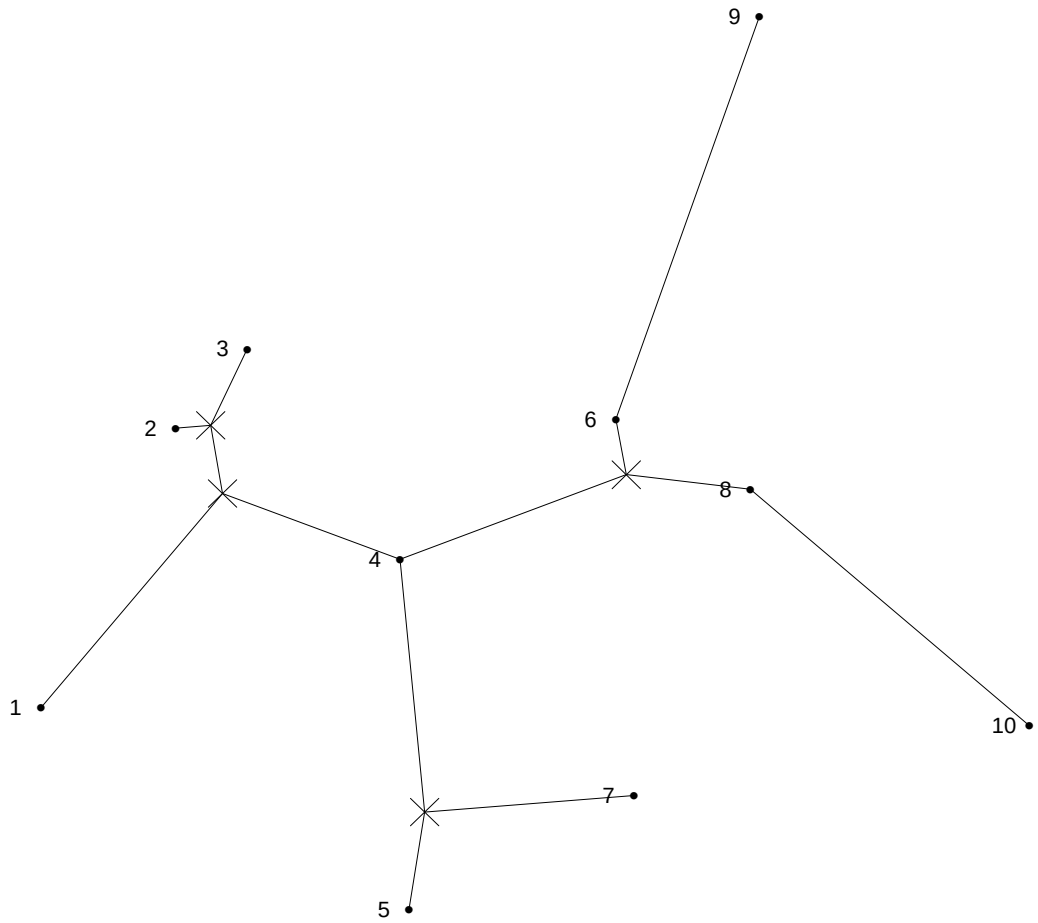


Figure C.6. Four S-points survive the removal one degree one and two points. Three rounds of this was necessary to move from the solution in Figure C.5 to this solution. Unfortunately none of the S-points are in their optimal location.

and B (in Figure C.7). Figure C.8 shows the result of adding the S-point. But now the three S-points A, B and C must be moved to their optimal positions. The five point FST with the particular topology for the points 4, 5, 6, 7 and 8 does exist. Figure C.9 shows the solution after moving the three S-points. This is the solution at the end of the third iteration.

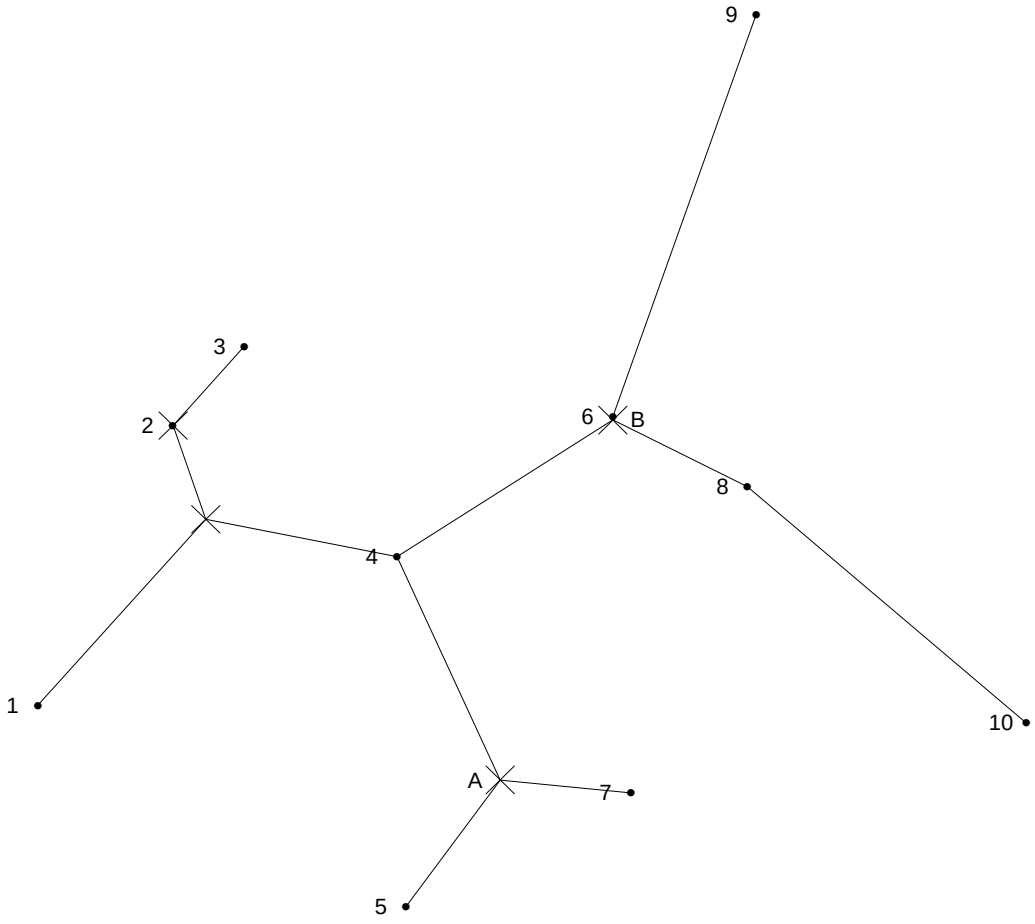


Figure C.7. The four S-points are moved to their optimal positions. S-point B is very close to point 6. This is the S-point for triangle (4,6,8).

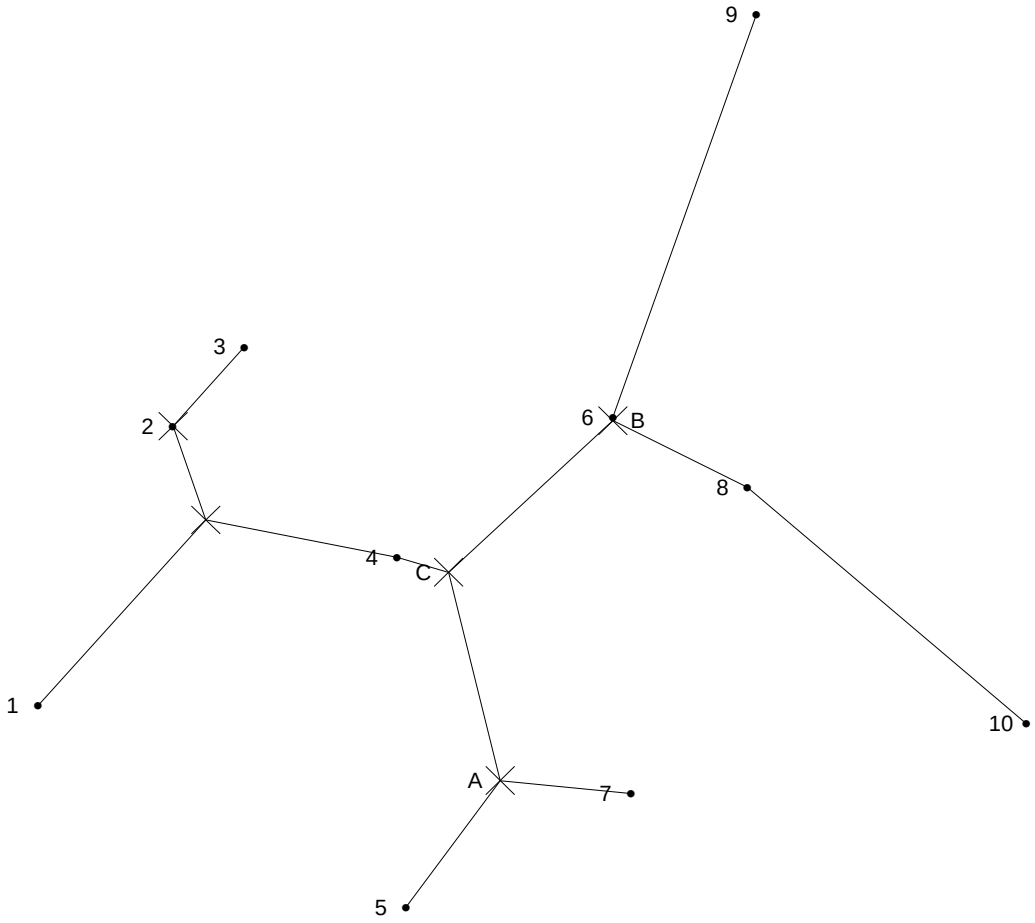


Figure C.8. S-point C is added to the triangle $(4, A, B)$ to eliminate an angle of less than 120°. But now S-points A, B and C are no longer in their optimal locations.

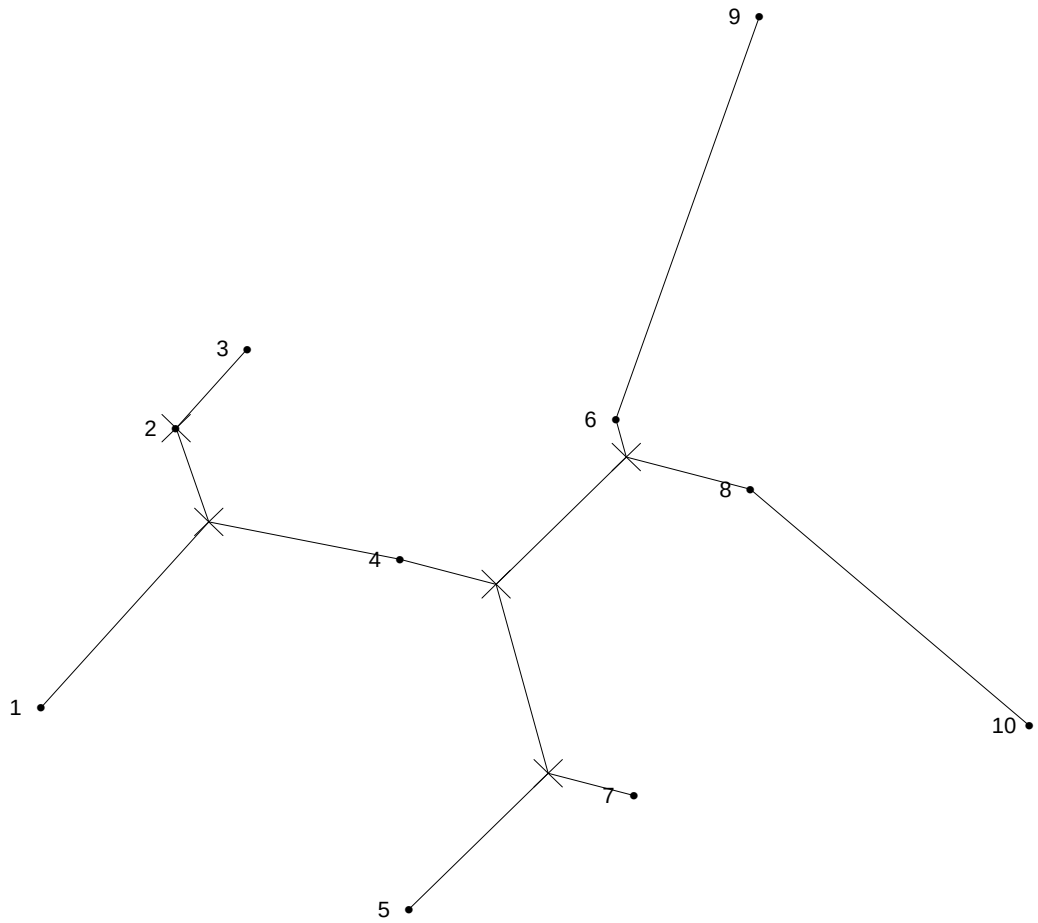


Figure C.9. The S-points of the FST of points 4, 5, 6, 7, and 8 are in their optimal locations. This is the solution at the end of the third iteration. It is longer than the solution at the beginning of the iteration shown in Figure C.4.

C.4 Fourth Iteration

The solution has changed therefore only one Delaunay triangulation is used this iteration. Figure C.10 shows the solution after removing all degree one and two S-points and moving points to their optimal positions.

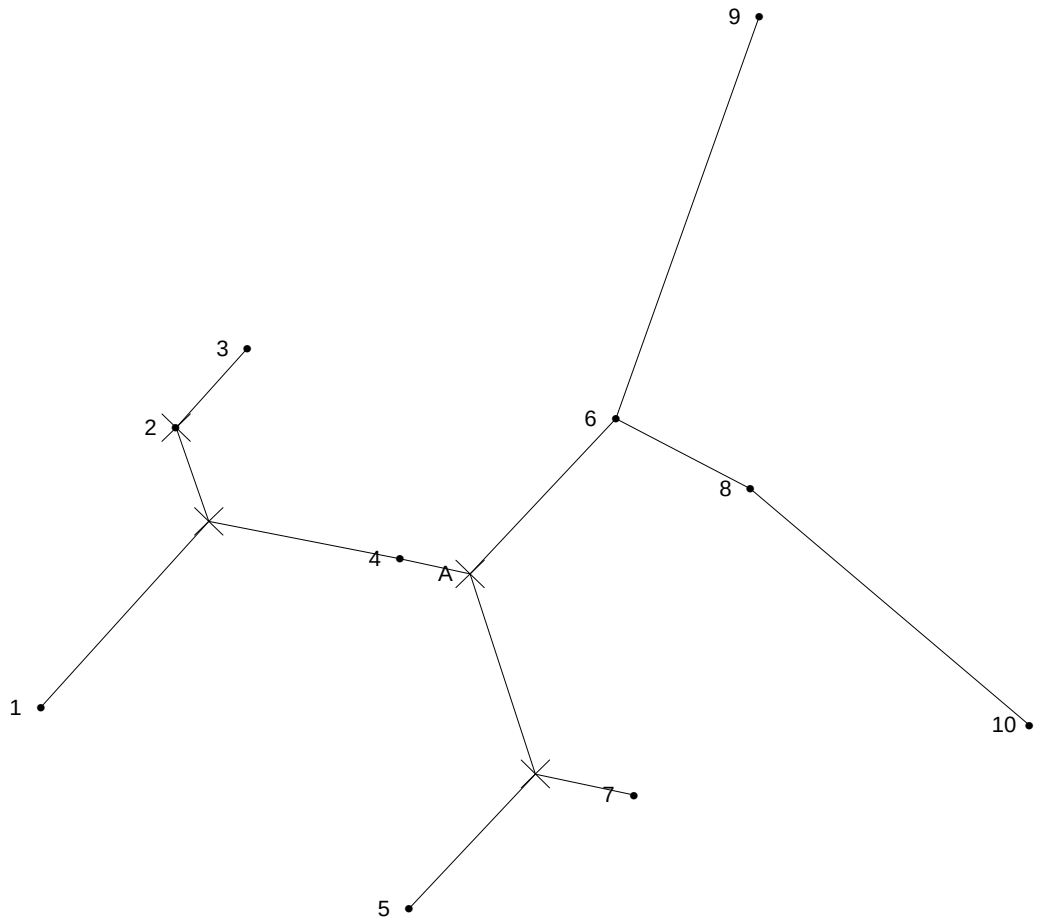


Figure C.10. All S-points are in their optimal locations. But there are two angles of less than 120°, both are at point 6.

The triangles $(A, 6, 8)$ and (6,8,9) both contain an angle of less than 120°. S-points are added to overcome this. Figure C.11 shows the undesirable result of doing this. However the remedy is straightforward, the S-point of triangle $(A, 6, 8)$ is removed because it is degree two. The solution is shown in Figure C.12. This is the solution at the end of the fourth iteration. It is different to the solution at the beginning of the iteration, and is longer than the best solution found to date.

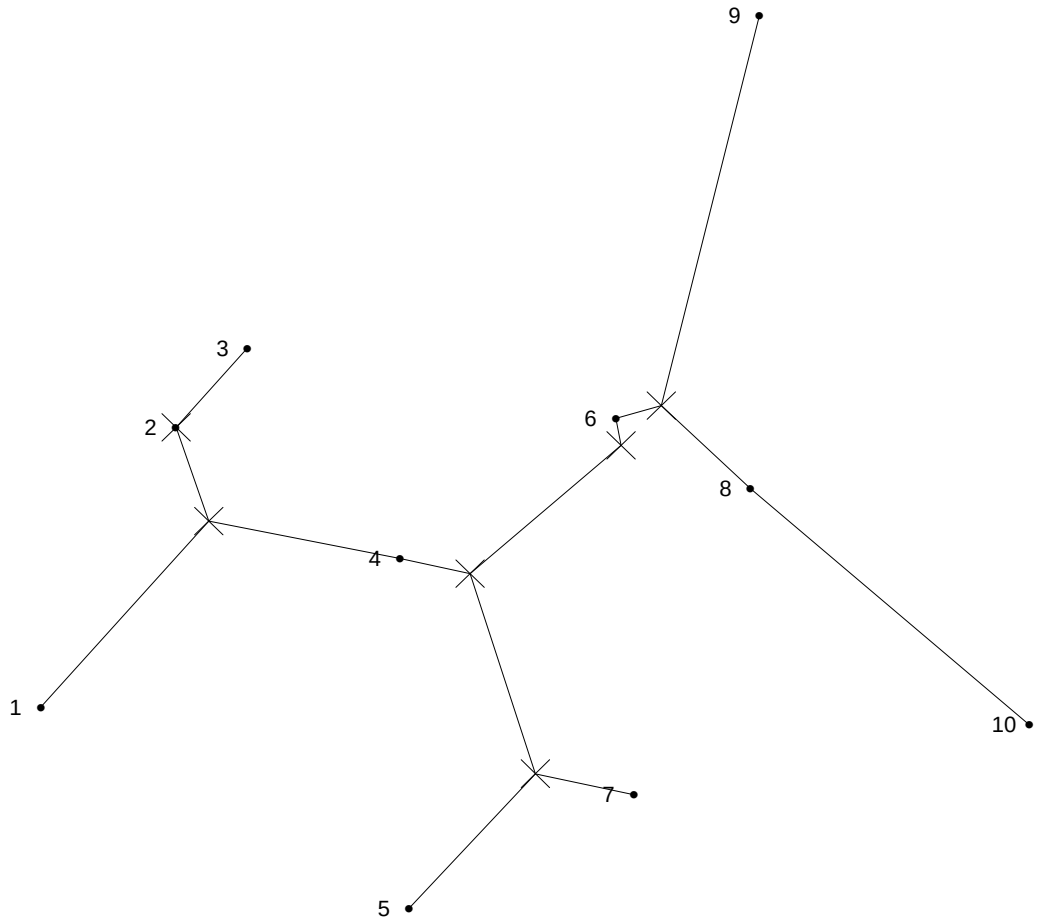


Figure C.11. One of the additional S-points adds nothing to the solution because it is degree two.

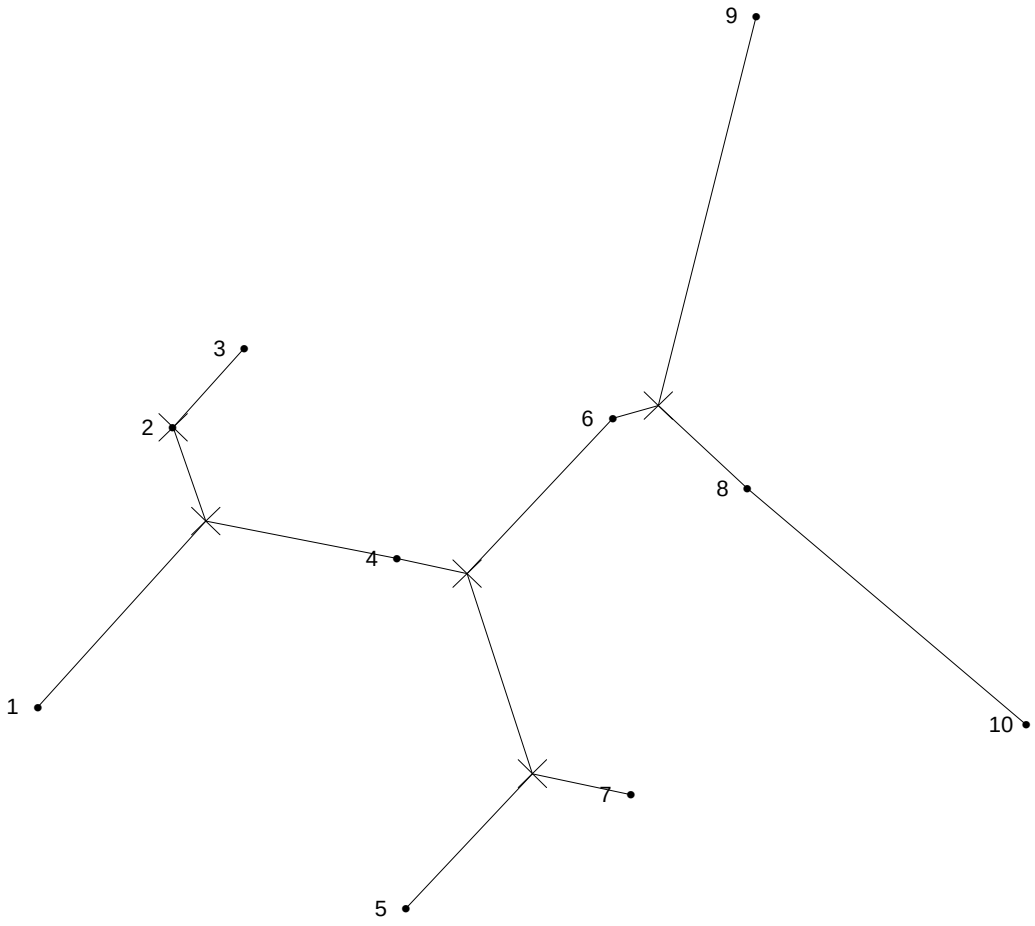


Figure C.12. The solution at the end of the fourth iteration.

C.5 Fifth Iteration

In this iteration one Delaunay triangulation is performed. The solution at the end of the iteration is identical to that at the beginning.

C.6 Sixth Iteration

The fifth iteration gave no change so two Delaunay triangulations are used in this iteration. Figure C.13 shows the solution after expansion and three rounds of removing degree one and two points. All the S-points are in non-optimal positions.

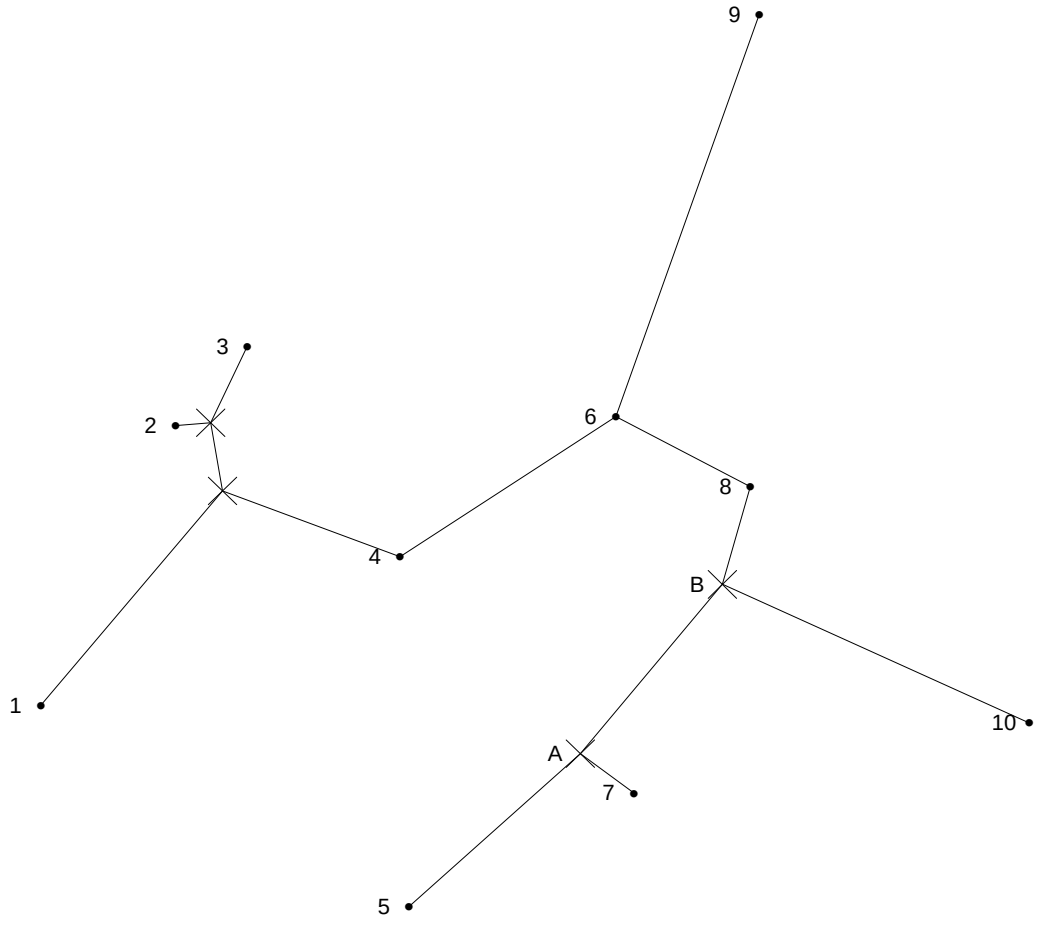


Figure C.13. Four S-points survive the degree one and two cull. All the S-points are not in optimal positions. The now familiar S-points of the 1, 2, 3 and 4 point FST and the new additions, A and B, connecting points 5, 7, 8 and 10 must be moved.

Attempting to move the S-points to their optimal locations reveals that S-point A can not exist, so is removed, and B becomes the S-point of triangle (7,8,10). The updated solution is shown in Figure C.14.

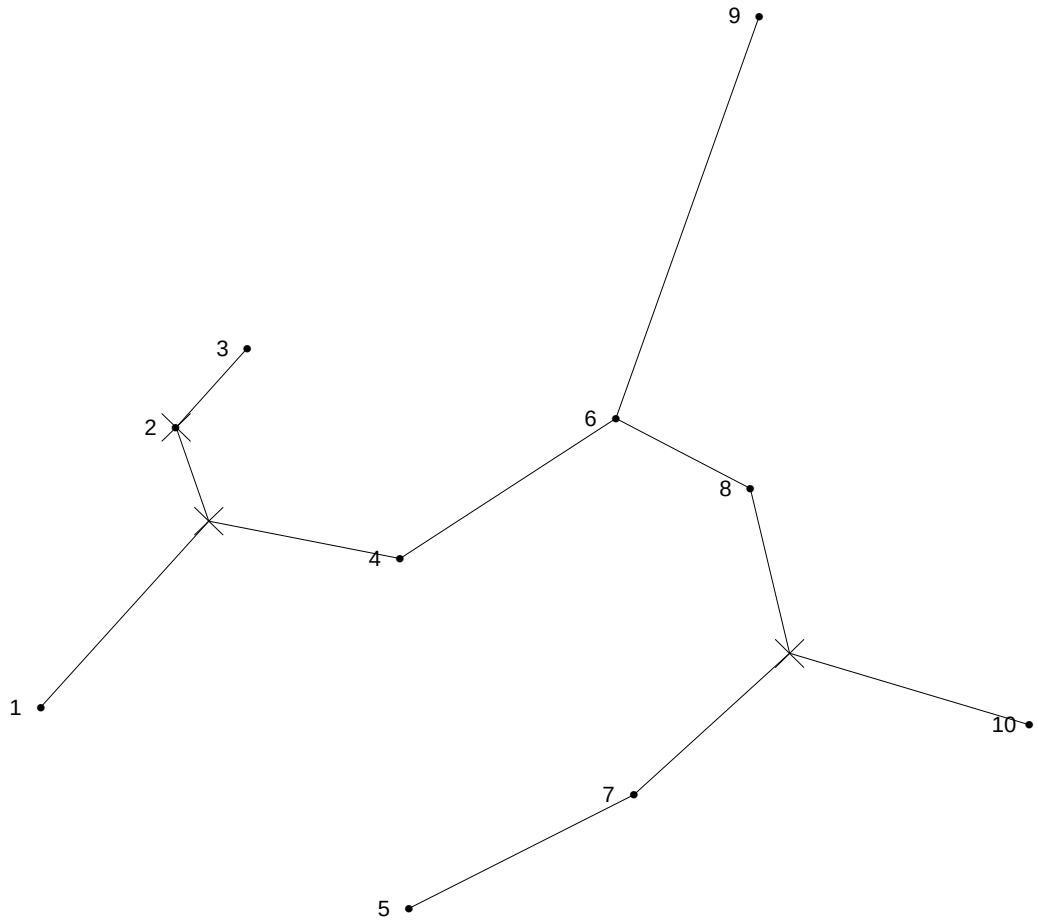


Figure C.14. The S-points are moved to their optimal positions. One of the four in Figure C.13 has been removed.

It is necessary to add, yet again, a S-point to the triangle (6,8,9) to remove an angle of less than 120°. Figure C.15 shows the new solution. This is the solution at the end of the iteration. It is different to that at the beginning of the iteration, but the same as at the end of the first iteration. At that point the number of expansions in the next iteration was also one. Therefore, there is a cycle of five iterations.

The solution shown in Figure C.15 is the solution produced by the heuristic. The annealing version of the heuristic is necessary to eventually break the cycle. The solution is the optimal solution to the ten point example problem.

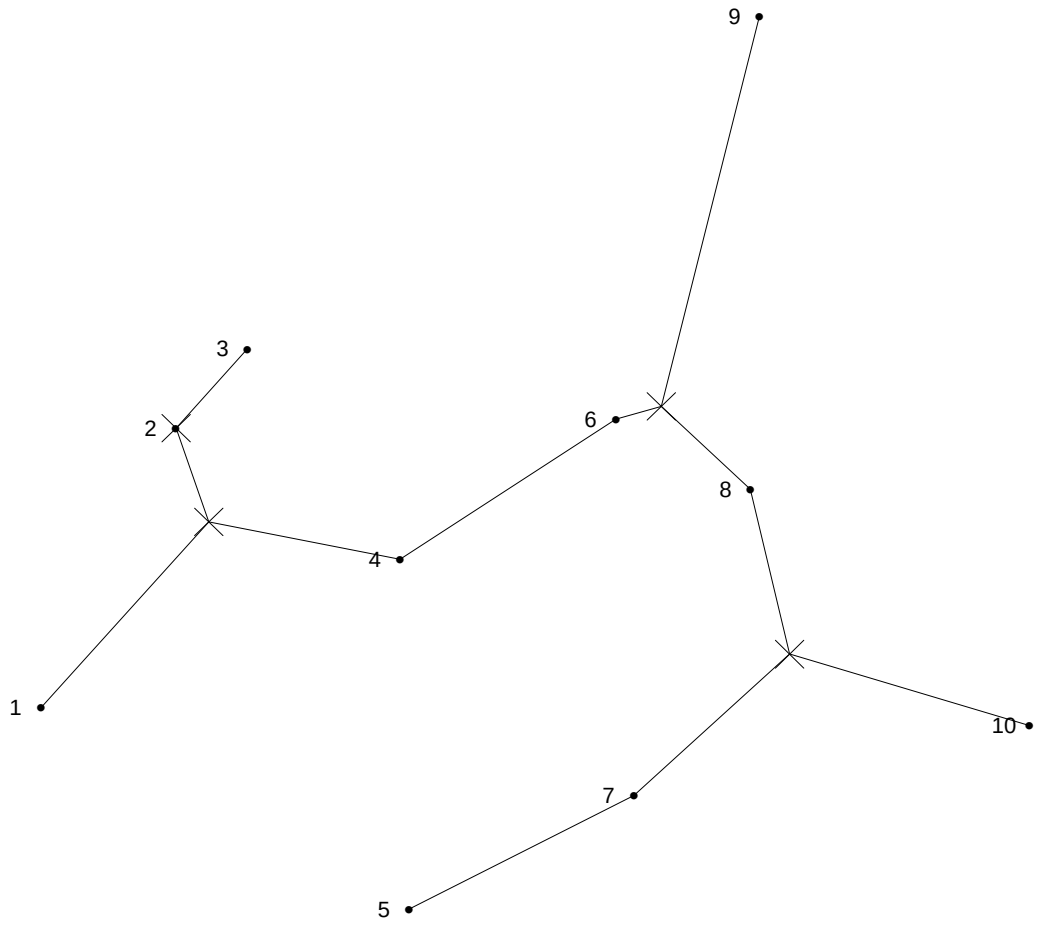


Figure C.15. The solution is the same as that at the end of the first iteration shown in Figure C.3. It is also the best solution found to date and the solution to the ten point example problem produced by the heuristic.

Appendix D

A 2-change Example

For a n city symmetric travelling salesman problem two different 2-changes can be applied to the same cyclic permutation and give the same tour. This is shown by example in this appendix.

Figure D.1 shows cyclic permutation $\{6, 3, 4, 7, 1, 2, 5\}$ on the left and the permutation $\{6, 4, 7, 3, 1, 2, 5\}$ on the right. The latter is the result of a 2-change with $l = 2$ and $m = 3$ applied to the former.

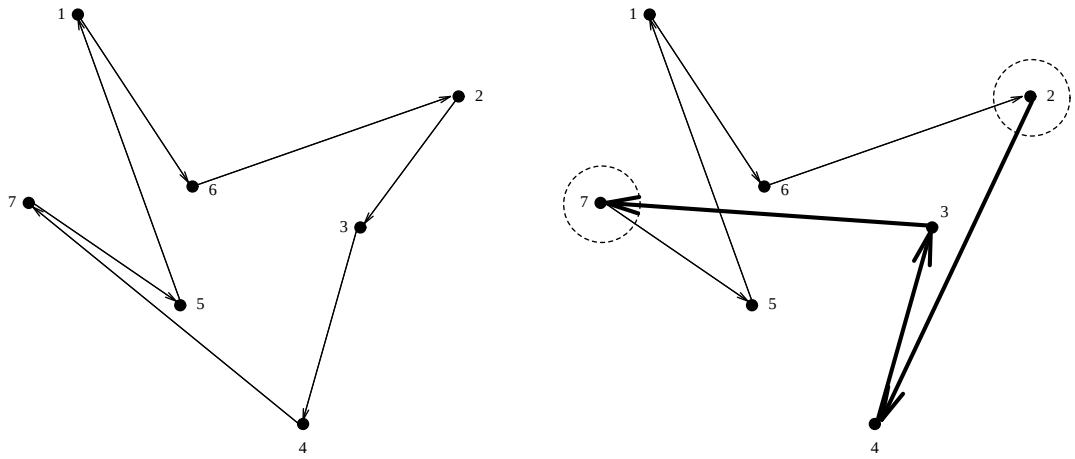


Figure D.1. The permutation on the right is the result of applying a 2-change with $l = 2$ and $m = 3$ to the permutation on the left.

Figure D.2 shows permutation $\{6, 3, 4, 7, 1, 2, 5\}$ on the left and the permutation $\{5, 6, 4, 2, 7, 1, 3\}$ on the right. The latter is the result of a 2-change with $l = 4$ and $m = 6$ applied to the former.

Both right hand side permutations in the figures represent the same tour in the symmetric problem. The difference is the direction of travel around the tour.

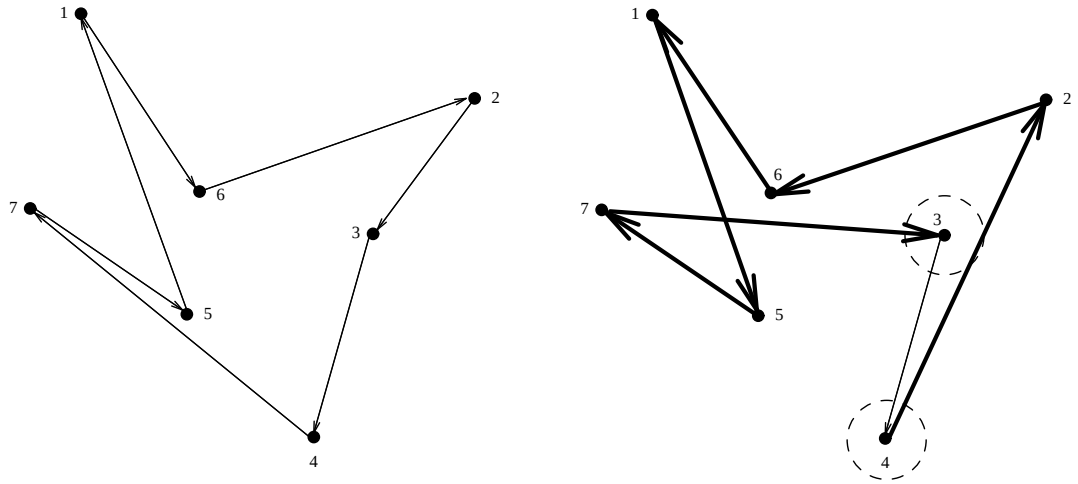


Figure D.2. The permutation on the right is the result of applying a 2-change with $l = 4$ and $m = 6$ to the permutation on the left.

Appendix E

The Finite Sequence of 2-changes Condition

One of the conditions for the asymptotic convergence of the simulated annealing algorithm to a globally minimal configuration is that it is possible to obtain any configuration in a finite number of transitions from any other configuration at any value of the control parameter. In this appendix the method of Laarhoven [25] for creating a finite sequence of 2-change transitions for the *symmetric* travelling salesman problem is described and demonstrated by example.[1]

For a problem with n cities it is possible to transform tour i into tour j by applying a sequence of n 2-changes to cyclic permutation π_i and subsequent permutations. Define the sequence of permutations $\{\pi_0, \ldots, \pi_n\}$ where $\pi_0 = \pi_i$ and $\pi_n = \pi_j$. Permutation π_t is obtained from π_{t-1} using a 2-change with $l = t$ and $m = m_t$ where m_t is given by

$$\pi_{t-1}^{m_t-1}(t) = \pi_j(t).$$

The 2-changes are such that after t transitions the first t cities in tour t, represented by π_t, are identical to the first t cities in tour j. The following example demonstrates the method.

The current eight city tour is described by permutation $\pi_i = \{5, 8, 2, 6, 4, 3, 1, 7\}$, and the desired tour is given by permutation $\pi_j = \{4, 3, 6, 2, 7, 5, 8, 1\}$ (see Figure E.1).[2] The tours already have three of eight edges in common, (7,8), (2,3) and (3,6).

The first 2-change requires that $\pi_0^{m_i-1}(1) = \pi_j(1) = 4$. Looking at π_i city 4 is the second city visited after city 1. Therefore $m_1 = 3$ and the 2-change is $l = 1$ and $m = 3$. Figure E.2 shows this 2-change and the new permutation $\pi_1 = \{4, 8, 2, 5, 6, 3, 1, 7\}$.

The second 2-change requires $\pi_1^{m_2-1}(2) = \pi_j(2) = 3$, $\pi_1^7(2) = 3$ therefore $m_2 = 8$. But $n = 8$ and $\pi_1^8(2) = 2$. Should all cities between 2 and 2 be visited in

[1] Laarhoven does not state whether the method works for all travelling salesman problems or just symmetric problems. It is conjectured it only works for symmetric problems because of the undirected nature of such problems. The implication of direction when using permutations to describe a tour possibly causes difficulties for asymmetric, or directed, problems.

[2] Reverse direction permutations could have been used.

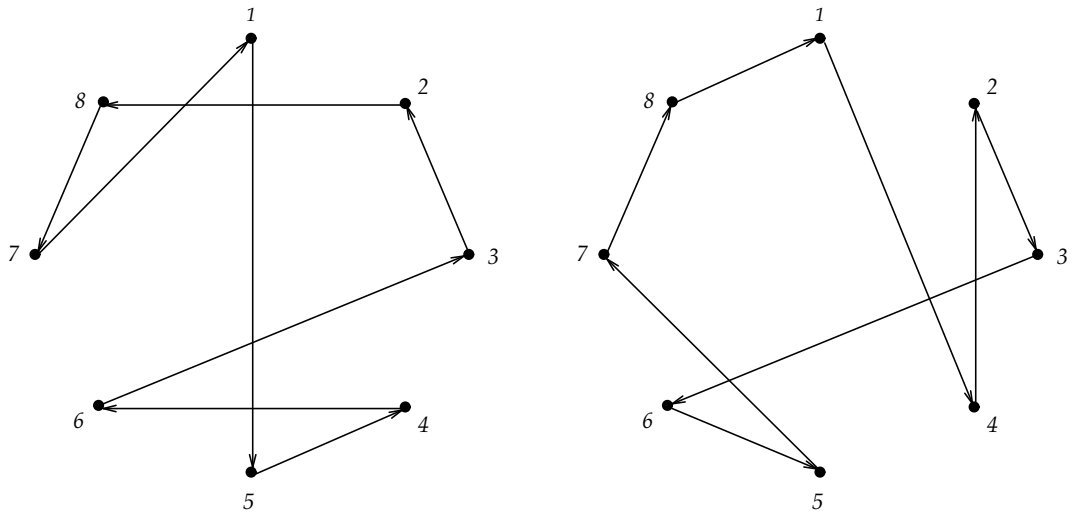


Figure E.1. On the left is the current tour, permutation $\{5, 8, 2, 6, 4, 3, 1, 7\}$. On the right is the desired tour, permutation $\{4, 3, 6, 2, 7, 5, 8, 1\}$.

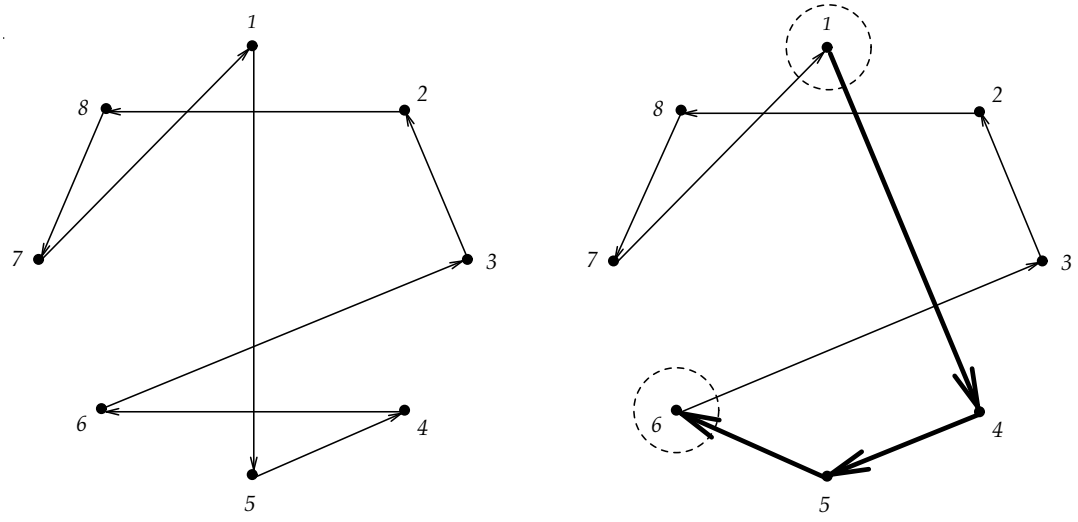


Figure E.2. The first 2-change: on the left is the current tour, permutation $\{5, 8, 2, 6, 4, 3, 1, 7\}$ and on the right is permutation $\{4, 8, 2, 5, 6, 3, 1, 7\}$, found using 2-change $l = 1$ and $m = 3$.

reverse direction, that is reverse the entire tour? Or, should this be thought of as "do no 2-change"? The first option causes problems in later 2-changes because the work of earlier transitions is destroyed. The second option can be justified by using the reverse permutation. The reverse permutation gives $m_2 = 2$ because city 3 is the immediate successor of 2 in the reverse permutation. A 2-change with $m = 2$ results in no change to the permutation. The unaltered reversed permutation is itself reversed to give the original permutation. If it is not reversed then the problems with the first option occurs.[3] The result is $\pi_2 = \pi_1$, but the first two elements of π_2 are not identical to the first two of π_j. However, if the tours represented by the permutations are considered instead of the permutations themselves then the edges (1,4) and (2,3) are in π_2 and π_j. This is the desired result of the method.

The third 2-change requires $\pi_2^{m_3-1}(3) = \pi_j(3) = 6$, $\pi_2^7(3) = 6$, and therefore $m_3 = 8$. The same situation as above! No 2-change is performed and $\pi_3 = \pi_2$.

The fourth 2-change requires $\pi_3^{m_4-1}(4) = \pi_j(4) = 2$, $\pi_3^4(4) = 2$, therefore $m_4 = 5$. Applying the 2-change with $l = 4$ and $m = 5$ to π_3 gives the permutation $\pi_4 = \{4, 3, 6, 2, 8, 5, 1, 7\}$ (see Figure E.3).

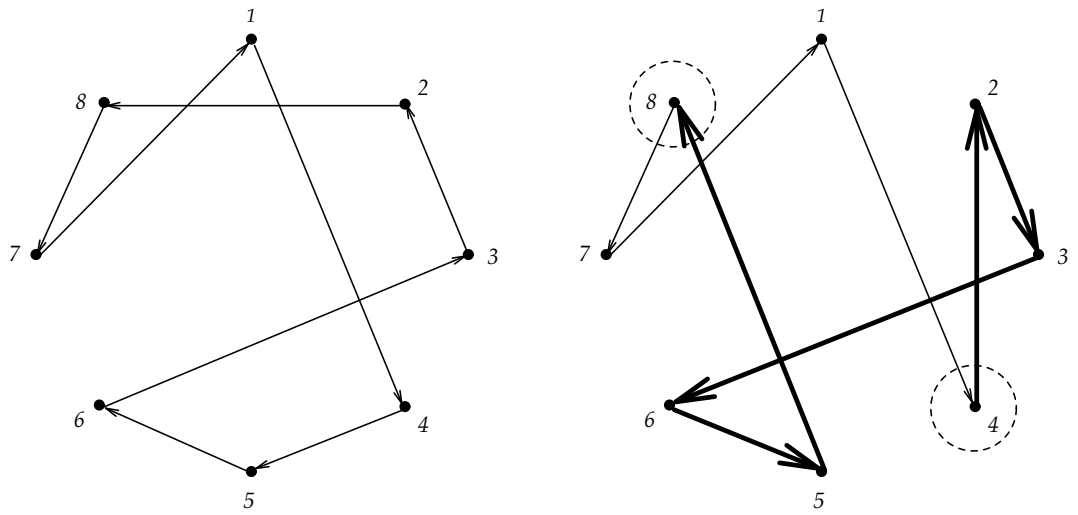


Figure E.3. The fourth 2-change: on the left is the current permutation $\{4, 8, 2, 5, 6, 3, 1, 7\}$ and on the right is permutation $\{4, 3, 6, 2, 8, 5, 1, 7\}$, found using 2-change $l = 4$ and $m = 5$.

The fifth 2-change requires $\pi_4^{m_5-1}(5) = \pi_j(5) = 7$, $\pi_4^2(5) = 7$, therefore $m_5 = 3$. Applying the 2-change with $l = 5$ and $m = 3$ to π_4 gives the permutation $\pi_5 = \{4, 3, 6, 2, 7, 5, 8, 1\}$ (see Figure E.4)

[3] It is this ability to reverse permutations for symmetric problems to achieve the desired tour, although the permutation may be for the tour in the reverse direction, that begs the question whether it is valid for asymmetric problems. In an asymmetric problem a tour and its reverse tour can have different costs, perhaps infinite. At any finite value of the control parameter a transition to an infinite cost configuration is impossible (for the purposes of circumventing the $m = n$ difficulty). Therefore for some configurations it is not possible to obtain any other configuration using 2-change transitions. This does not mean simulated annealing will not find good solutions for asymmetric problems just that asymptotic convergence is no longer guaranteed.

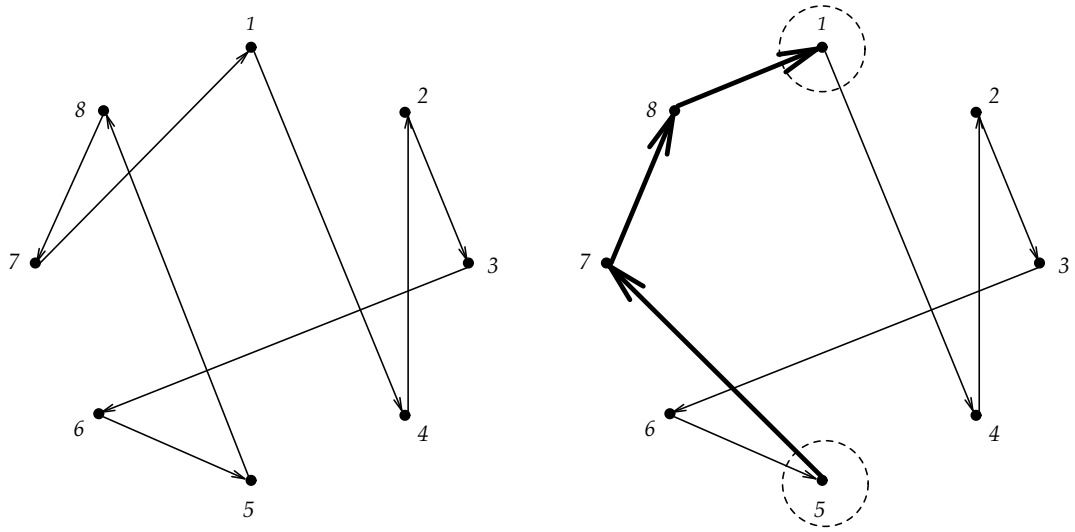


Figure E.4. The fifth 2-change: on the left is the current permutation $\{4, 3, 6, 2, 8, 5, 1, 7\}$ and on the right is permutation $\{4, 3, 6, 2, 7, 5, 8, 1\}$, found using 2-change $l = 5$ and $m = 3$.

The remaining three 2-changes give no further changes in the permutation, $\pi_8 = \pi_7 = \pi_6 = \pi_5$. The tour represented by π_8 is identical to that represented by π_j. In this case the permutations are identical, but this isn't always necessarily so. The tours are identical and this is the desired outcome.

Appendix F

Listing of the Travelling Salesman Problem Simulated Annealing Program

This appendix is a listing of the C++ program for finding the solution of a Euclidean travelling salesman problem using simulated annealing. A polynomial cooling schedule is used. The file containing **main()** is Main.cc.[1]

Table F.1 shows the necessary inputs to the program in the order they must be presented. The different outputs produced by different output level settings are shown in Table F.2. If the program is compiled with **-DGRAPHICS** in the compiler command a LEDA graphical window is displayed showing details of the annealing process in the form of the evolving tour. As with normal text output different types of graphical output are produced, and are explained in Table F.3.

[1]The preamble to the program listing in Appendix A describes the software and hardware used for all programs.

Input	Description
Output Level	An integer specifying the amount and type of output.
Annealer Seed	An integer seed for the acceptance test generator of the annealer.
Initial Acceptance Ratio	The initial acceptance parameter ϕ_0 of the polynomial cooling schedule.
Distance Parameter	The distance parameter δ of the polynomial cooling schedule.
Stopping Parameter	The stopping parameter ϵ_s of the polynomial cooling schedule.
Not used	An integer no longer used but still required for backward compatibility with earlier versions of the program and input data sets.
Smoothing Length	An integer specifying the number of chains over which to smooth the average cost in a chain. The smoothed average in used in determining the stopping ratio, which is compared to the stopping parameter.
Number of Points	The number of "cities" for which to find a tour.
Problem Seed	An integer seed for the 2-change operation.
Point Coordinates	The coordinates of the points.

Table F.1. A description of the input to the travelling salesman problem simulated annealing program.

Output Level	Description
$= -1$	For every transition the control parameter value and cost of the current configuration are output
≥ 1	The CPU time, the final value of the control parameter, the value of the stopping ratio, the cost of final configuration, and the configuration itself are output.
≥ 3	As for 1, and if the program was compiled using **-DBEST** the best cost and configuration found at any stage are output at the end of the annealing.
≥ 5	As for 3, and at the and of each chain the following are output: the chain number, the control parameter value, the average cost of configurations found in the chain, the standard deviation of costs, the value of the next control parameter, the smoothed average cost and the stop ratio.
≥ 10	As for 5, and • the point coordinates and distance matrix are output before annealing begins; • the schedule parameters are output before annealing begins; • for each iteration of the preliminary chain to find the initial control parameter value the following are output: the iteration number, the control parameter value, the number of decreasing cost transitions, the number of proposed increasing cost transitions, the average proposed increased cost and the next control parameter value; • for each transition of the annealing proper the following are output: the iteration number, the control parameter value, the change in cost, the value of the Metropolis criterion, the acceptance test random number, a zero if rejected or a one if accepted, the 2-change l and m, the current configuration and cost, and finally the neighbour configuration and cost.

Table F.2. A description of the text output produced by the travelling salesman problem simulated annealing program.

Output Level	Description
≥ 1	The control parameter value, current configuration cost, average cost of configurations found in the final chain and standard deviation of costs are displayed together with the final configuration.
≥ 3	As for 1, and if the program was compiled using **-DBEST** the best cost and configuration found at any stage is displayed at the end of the annealing.
≥ 5	As for 3, and at the end of each chain the current and best configurations are displayed.
≥ 10	As for 5, and for each transition of the annealing proper the following are displayed: the current configuration, the proposed configuration, their costs, the difference and whether or not the transition is accepted.

Table F.3. A description of the graphical output produced by the travelling salesman problem simulated annealing program.

F.1 Makefile

```
#This Makefile takes the .cc files defined by SOURCES and compiles and links
#then to give the program with name defined by PROGRAM.

LEDA = /u/grads/geoff/LEDA

#LEDA is a handy library of graph, list, array and geometrical objects.

X11 = /usr/openwin/lib

TARGET = TSP
SOURCES = Main Annealer TSPProblem

OBJECTS = $(SOURCES:%=%.o)

#OBJECTS is a list of .o filenames corresponding to SOURCES.

.SILENT:

#To see all that is happening comment out the .SILENT line. The horrific
#commands that do the compiling and linking will be shown.

.SUFFIXES: .cc .o
.KEEP_STATE:

CCC = g++

#g++ is the GNU C++ compiler and linker.

INCLUDES= -I$(LEDA)/incl

#INCLUDES is the directory in which to search for LEDA .h files.

CCFLAGS = -O2 -DBEST -DTRACE -DGRAPHICS

#-O2 is the optimisation flag for the g++ compiler and linker.

CPPFLAGS=
LDFLAGS =
LEDALIBS= -L$(LEDA) -lP -lG -lL -lWx

#LEDALIBS is the directory and list of library archive files in which to
#for LEDA objects and programs. -lP means libP.a etc.

X11LIBS = -L$(X11) -lxview -lolgx -lX11

COMPILE = $(CCC) $(CCFLAGS) $(INCLUDES) $(CPPFLAGS)-c

#COMPILE is the compile command without linking.

LINK = $(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS)

#LINK is the link command

.cc.o:
```

```
	@echo Compiling -- $*
	$(COMPILE) $<

#.cc.o means for each .cc with an old .o or no .o perform the compile command.

$(TARGET): $(OBJECTS)
	@echo Linking --- $@
	$(LINK)-o $(TARGET) $(OBJECTS) $(LEDALIBS) $(X11LIBS) -lm

#(TARGET): $(OBJECTS) means link any new .o files to give a new PROGRAM file
#or link all .o files to give a PROGRAM file if no PROGRAM file exists.
```

F.2 Annealer.h

```
#ifndef ANNEALER_H
#define ANNEALER_H

// The annealer class that uses Laarhoven's polynomial schedule.

#include <LEDA/basic.h>
#include <LEDA/list.h>

#include "Problem.h"
#include "UniformGenerator.h"

class Annealer{

 public:
  Annealer();
  ~Annealer();
  void solve(Problem& p);

 private:
  void oneStep();
  void getFirstC();
  void getNextC();
  void traceAnnealer();
  void traceOneStep();
  void traceSolve();
  void traceGetFirstC();
  void traceGetNextC();

#ifdef GRAPHICS
  void graphicsGetNextC();
  void graphicsOneStep();
#endif

  // the control parameter
  double c;
  // the problem
  pProblem P;
  // the acceptance random number generator
  pUniformGenerator URNG;
  // the CPU time
  float time;
```

```
  // the initial acceptance ratio
  double phi0;
  // the distance parameter
  double delta;
  // the stopping parameter
  double epsilon;
  // the number of chains over which to smooth the average cost in a chain
  int smoothingLength;
  // not used
  int m0;
  // the chain length
  int L;
  // the chain counter
  int chain;
  // the variable for testing whether to stop annealing
  bool stop;
  // the standard deviation of costs in a chain
  double sigma;
  // the list of average cost in a chain
  list<double> mulist;
  // the smoothed average cost
  double smu;
  // the previous smoothed average cost
  double lastsmu;
  // the average cost in a chain
  double mu;
  // the very first chain's average cost
  double mu1;
  // the previous control parameter
  double lastc;
  // the stopping ratio
  double stopRatio;
  // the number of decreasing cost moves for determining the first
  // value of the control parameter
  int m1;
  // the number of increasing cost moves
  int m2;
  // the sum of proposed increasing cost moves
  double sumIncreases;
  // the random number for testing acceptance of a move
  double t;
  // the acceptance probability
  double x;
  // the counter of transitions in a chain
  int i;
};

#endif
```

F.3 Annealer.cc

```
#include "Annealer.h"

#include "Trace.h"
```

```
// constructor
Annealer::Annealer(){
  // read the seed for the acceptance test generator
  int seed=read_int();
  URNG=new UniformGenerator(seed);
  // read the schedule parameters
  phi0 = read_real();
  delta = read_real();
  epsilon = read_real();
  m0 = read_int();
  smoothingLength = read_int();
  // initialise annealing variables
  chain = 1;
  c=10000;
  smu=0;
  stopRatio=0;
  traceAnnealer();
}

// destructor
Annealer::~Annealer(){
  delete URNG;
}

// solve the problem
void Annealer::solve(Problem& p){
  time=used_time();
  P=&p;
  // get the chain length from the problem
  L=P→getL();
  // get the first control parameter value
  getFirstC();
  // anneal
  do{
    // reset chain statistics
    double sumCosts=0;
    double sumCostsSquared=0;
    // do a chain
    loop(i,1,L){
      // attempt a move
      oneStep();
      // collect cost information
      double cost=P→getSolutionDistance();
      sumCosts+=cost;
      sumCostsSquared+=cost*cost;
    }
    // calculate statistics
    mu=sumCosts/L;
    double var=sumCostsSquared/L-mu*mu;
    if (var<0) sigma=0;
    else sigma=sqrt(var);
    mulist.append(mu);
    // catch the first chain's average cost
    if (chain==1) mu1=mu;
    // calculate smoothed average values
    if (chain≥smoothingLength){
      double m;
```

```
      double sum=0;
      forall(m,mulist) sum+=m;
      lastsmu=smu;
      smu=sum/smoothingLength;
      mulist.pop();
    }
    // find the next value of the control parameter
    getNextC();
    // stop annealing when the control parameter is zero or
    // the stopping rule using the stopping parameter is satisfied
  } while (!stop);
  traceSolve();
}

void Annealer::oneStep(){
  bool changed=false;
#ifdef GRAPHICS
  if (trace≥10){
    Wp→clear();
    P→plotSolution();
  }
#endif
  // generate a neighbour configuration
  P→generateNeighbour();
#ifdef GRAPHICS
  graphicsOneStep();
#endif
  // accept or reject the move using the Metropolis criterion
  if (P→getChange()>1E-7){
    t=URNG→rand();
    x=exp(-P→getChange()/c);
    if (t<x){
      P→updateSolution();
      changed=true;
    }
  }
  else{
    t=1;
    x=1;
    if (P→getChange()≤-1E-7){
      P→updateSolution();
      changed=true;
      t=0;
    }
  }
#ifdef TRACE
  traceOneStep();
#endif
#ifdef GRAPHICS
  if (trace≥10){
    if (t<x){
      Wp→draw_text(0,Wp→ymax()-0.1,
                   string("%24.4f Accept", P→getChange()));
    }
    else{
      Wp→draw_text(0,Wp→ymax()-0.1,
                   string("%24.4f Reject", P→getChange()));
```

```
    }
    wait(2*waitingtime);
  }
#endif
  // If the move was not accepted undo any changes
  if (!changed)
    P→undoChanges();
}

// find the first value of the control parameter
void Annealer::getFirstC(){
  m1=0;
  m2=0;
  lastc=c;
  sumIncreases=0;
  // attempt L transitions
  loop(i,1,L){
    oneStep();
    if (P→getChange()>0){
      ++m2;
      sumIncreases+=P→getChange();
    }
    else{
      ++m1;
    }
    if (m2>0){
      lastc=c;
      // update the value of the control parameter
      c=(sumIncreases/m2)/log(m2/(m2*phi0-m1*(1-phi0)));
    }
    traceGetFirstC();
  }
}

// get the next value of the control parameter
void Annealer::getNextC(){
  stop=false;
  if (chain>smoothingLength){
    // calculate the stopping ratio
    stopRatio=(c/mu1)*(smu-lastsmu)/(c-lastc);
    stop=fabs(stopRatio)<epsilon;
  }
  lastc=c;
  // calculate the next value of the control parameter
  c=c/(1+c*log(1+delta)/(3*sigma));
  // check if the value is zero
  stop|=c<1E-7;
  traceGetNextC();
#ifdef GRAPHICS
  graphicsGetNextC();
#endif
  ++chain;
}

void Annealer::traceAnnealer(){
  if (trace≥10) cout ≪ string("%10.7f%10.7f%10.7f%6d\n",
                              phi0,delta,epsilon,smoothingLength);
```

```
}

void Annealer::traceOneStep(){
  if (trace==-1){
    cout ≪ string("%21.7f%15.7f\n", c, P→getSolutionDistance());
  }
  if (trace≥10){
    cout ≪ string("%21.7f%15.7f", c, P→getChange());
    cout ≪ string("%12.4f%15.4f", t, x);
    if (t<x){
      cout ≪ "  1\n";
    }
    else{
      cout ≪ "  0\n";
    }
  }
}

void Annealer::traceSolve(){
  if (trace≥1){
    cout ≪ string("%21.7f%15.7f%15.7f\n", used_time(time), c,
                  stopRatio);
    cout ≪ string("%21.7f", P→getSolutionDistance());
    P→printSolution();
    newline;
  }
#ifdef BEST
  if (trace≥3){
    cout ≪ string("%21.7f", P→getBestDistance());
    P→printBest();
    newline;
  }
#endif
}

void Annealer::traceGetFirstC(){
  if (trace≥10){
    cout ≪ string("%6d%15.7f%7d%8d", i, lastc, m1, m2);
    if (m2>0){
      cout ≪ string("%15.7f%15.7f\n", sumIncreases/m2, c);
    }
    else{
      cout ≪ string("%15.7f%15.7f\n", 0.0, c);
    }
  }
}

void Annealer::traceGetNextC(){
  if (trace≥5){
    cout ≪ string("%6d%15.7f%15.7f", chain, lastc, mu);
    cout ≪ string("%15.7f%15.7f%15.7f", sigma, c, smu);
    cout ≪ string("%15.7f\n", stopRatio);
  }
}

#ifdef GRAPHICS
void Annealer::graphicsGetNextC(){
```

```
  if (trace≥5){
    Wp→clear();
    P→plotSolution(black);
    P→plotBest();
    Wp→draw_text(0,Wp→ymax()-0.1,
                 string("%8.4f%9.4f%9.4f%9.4f",
                        c,P→getSolutionDistance(),
                        mu,sigma));
    Wp→flush();
  }
}

void Annealer::graphicsOneStep(){
  if (trace≥10){
    P→plotChange();
    wait(2∗waitingtime);
    P→plotSolution(blue);
    Wp→flush();
  }
}
#endif
```

F.4 Generator.h

```
#ifndef GENERATOR_H
#define GENERATOR_H

// An integer uniform random number generator using the GNU g++
// library random number generators.

#include <ACG.h>
#include <RndInt.h>

class Generator{
private:
  ACG∗ pRNG;
  RandomInteger∗ pRnd;
public:
  Generator(int low=1, int high=1, int seed=1){
    pRNG = new ACG(seed, 30);
    pRnd = new RandomInteger(low, high, pRNG);
  }
  ∼Generator(){
    delete pRNG;
    delete pRnd;
  }
  long rand(){
    return pRnd→asLong();
  }
  long operator()(){
    return rand();
  }
};
typedef Generator ∗pGenerator;
```

```
#endif
```

F.5 Main.cc

```
#include "Trace.h"
#include "Annealer.h"
#include "TSPProblem.h"

int trace;

#ifdef GRAPHICS
window* Wp;
#endif

main(){

  trace=read_int();

#ifdef GRAPHICS
  window W(600,625);
  W.init(-0.1,1.1,-0.1);
  W.set_show_coordinates(false);
  W.set_node_width(3);
  W.set_mode(src_mode);
  Wp=&W;
#endif

  Annealer A;
  TSPProblem P;

  A.solve(P);

#ifdef GRAPHICS
  W.acknowledge(" Done!  ");
#endif

}
```

F.6 Problem.h

```
#ifndef PROBLEM_H
#define PROBLEM_H

// A general problem class for use with the annealer defined in
// Annealer.h. This class is the base class for all problems.

#ifdef GRAPHICS
#include <LEDA/window.h>
#endif

class Problem;
typedef Problem* pProblem;
```

```
class Problem{
protected:
  double solutionDistance;
  double neighbourDistance;
  double bestDistance;
  double change;
public:
  Problem(){
    solutionDistance=0;
    neighbourDistance=0;
    bestDistance=0;
    change=0;
  }
  virtual ~Problem(){};
  virtual void generateNeighbour(){};
  virtual void updateSolution(){};
  virtual void printSolution(){};
  virtual void printBest(){};
  virtual void undoChanges(){};
#ifdef GRAPHICS
  virtual void plotSolution(color c=black){};
  virtual void plotBest(){};
  virtual void plotChange(){};
#endif
  virtual int getL(){
    return 0;
  }
  double getSolutionDistance(){
    return solutionDistance;
  }
  double getNeighbourDistance(){
    return neighbourDistance;
  }
  double getBestDistance(){
    return bestDistance;
  }
  double getChange(){
    return change;
  }
};

#endif
```

F.7 TSPProblem.h

```
#ifndef TSPPROBLEM_H
#define TSPPROBLEM_H

// The Travelling Salesman Problem class.

#include <LEDA/array.h>
#include <LEDA/point.h>

#include "Generator.h"
#include "Problem.h"
```

```
typedef array<int> TSPSolution;
typedef TSPSolution *pTSPSolution;

typedef array<point> Points;
typedef Points *pPoints;

typedef array2<double> DistanceMatrix;
typedef DistanceMatrix *pDistanceMatrix;

class TSPProblem : public Problem{

public:
  TSPProblem();
  ~TSPProblem();
  void generateNeighbour();
  void updateSolution();
  int getL();
  void printSolution();
  void printNeighbour();
  void printBest();

#ifdef GRAPHICS
  void plotSolution();
  void plotBest();
  void plotPoints();
#endif

private:
  // problem size
  int n;
  // random numbers for the 2-change
  int l, m;
  // array of point coordinates
  pPoints positions;
  // array of distances between points
  pDistanceMatrix distances;
  // current configuration
  pTSPSolution solution;
  // neighbour configuration
  pTSPSolution neighbour;
  // best configuration found to date
  pTSPSolution best;
  // l generator
  pGenerator lRNG;
  // m generator
  pGenerator mRNG;
  // array used in 2-change
  array<bool>* done;
  // cost function
  double distance(TSPSolution& s);
  void newNeighbour();
  void traceTSPProblem();
  void traceGenerateNeighbour();
};

typedef TSPProblem *pTSPProblem;
```

```
#endif
```

F.8 TSPProblem.cc

```
#include "TSPProblem.h"

#include <LEDA/basic.h>
#include <LEDA/point.h>

#include "Trace.h"

//constructor
TSPProblem::TSPProblem(){
  // read the number of points
  n=read_int();
  // read the seed for the l and m generators
  int seed=read_int();
  positions=new Points(1,n);
  distances=new DistanceMatrix(1,n,1,n);
  solution=new TSPSolution(1, n);
  neighbour=new TSPSolution(1, n);
  best=new TSPSolution(1, n);
  lRNG=new Generator(1, n, seed);
  mRNG=new Generator(3, n-1, seed);
  done=new array<bool>(1,n);
  TSPSolution &s = *solution;
  TSPSolution &b = *best;
  DistanceMatrix &d = *distances;
  Points &p = *positions;
  int i;
  // create an intial configuration equal to 2,3,...n,1
  loop(i, 1, n-1){
    s[i] = i+1;
    b[i] = i+1;
  }
  s[n] = 1;
  b[n] = 1;
  // read the point coordinates
  p.read();
  int j;
  // calculate the distances between all points
  loop(i, 1, n){
    loop(j, 1, n){
      d(i, j) = p[i].distance(p[j]);
    }
  }
  // find the current configuration cost
  solutionDistance = distance(s);
  bestDistance = solutionDistance;
  traceTSPProblem();
}

//destructor
TSPProblem::~TSPProblem(){
```

```
  delete solution;
  delete neighbour;
  delete best;
  delete distances;
  delete lRNG;
  delete mRNG;
}

//2-change
void TSPProblem::generateNeighbour(){
  l = lRNG→rand();
  m = mRNG→rand();
  TSPSolution &s = *solution;
  DistanceMatrix &d = *distances;
  //find the change in cost
  register int i;
  register piMminusOne = s[l];
  loop(i, 2, m-1) piMminusOne = s[piMminusOne];
  register int piM = s[piMminusOne];
  change = d(l, piMminusOne) + d(s[l], piM) - d(l, s[l]) - d(piMminusOne, piM);
#ifdef TRACE
  traceGenerateNeighbour();
#endif
}

void TSPProblem::updateSolution(){
  //find the neighbour tour
  newNeighbour();
  // calculate the neighbour's cost
  solutionDistance = distance(*neighbour);
  // make the neighbour the new current tour
  pTSPSolution s = solution;
  solution = neighbour;
  neighbour = s;
#ifdef BEST
  //update the best tour if necessary
  if (solutionDistance<bestDistance){
    TSPSolution &b = *best;
    TSPSolution &s = *solution;
    register int i;
    loop(i,1,n) b[i] = s[i];
    bestDistance = solutionDistance;
  }
#endif
}

int TSPProblem::getL(){
  return n*(n-3)/2;
}

void TSPProblem::printSolution(){
  solution→print();
}

void TSPProblem::printNeighbour(){
  neighbour→print();
}
```

```
void TSPProblem::printBest(){
  best→print();
}

inline double TSPProblem::distance(TSPSolution& s){
  double t = 0;
  DistanceMatrix &d = *distances;
  register int i;
  loop(i, 1, n) t += d(i, s[i]);
  return t;
}

inline void TSPProblem::newNeighbour(){
  TSPSolution &s = *solution;
  TSPSolution &t = *neighbour;
  array<bool> &d = *done;
  register int i, j;
  //find the neighbour configuration given l and m
  loop(i, 1, n) d[i] = false;
  t[l] = l;
  loop(i, 2, m){
    t[l] = s[t[l]];
    t[s[l]] = s[t[l]];
  }
  d[l] = true;
  d[s[l]] = true;
  j = s[l];
  loop(i, 2, m-1){
    t[s[j]] = j;
    d[s[j]] = true;
    j = s[j];
  }
  loop(i, 1, n){
    if (!d[i]) t[i] = s[i];
  }
}

void TSPProblem::traceTSPProblem(){
  if (trace≥10){
    Points &p=*positions;
    DistanceMatrix &d=*distances;
    int i, j;
    loop(i, 1, n) cout ≪ string("%6d%15.7f%15.7f\n",
                                i, p[i].xcoord(),p[i].ycoord());
    loop(i, 1, n){
      loop(j, 1, n){
        if (i<j){
          cout ≪ string("%6d%6d%15.7f\n", i, j, d(i,j));
        }
      }
    }
  }
}

void TSPProblem::traceGenerateNeighbour(){
  if (trace≥10){
```

```
    newNeighbour();
    neighbourDistance = distance(*neighbour);
    cout << string("%21.7f", solutionDistance);
    cout << string("%7d%8d ", l, m);
    printSolution();
    cout << string("\n%21.7f%15.7f ", neighbourDistance, change);
    printNeighbour();
    newline;
  }
}

#ifdef GRAPHICS
void TSPProblem::plotPoints(){
  Points &p=*positions;
  register int i;
  loop(i,1,n) Wp->draw_filled_node(p[i]);
}

void TSPProblem::plotSolution(){
  plotPoints();
  register int i;
  TSPSolution &s=*solution;
  Points &p=*positions;
  loop(i,1,n) Wp->draw_segment(p[i],p[s[i]]);
}

void TSPProblem::plotBest(){
  register int i;
  TSPSolution &b=*best;
  Points &p=*positions;
  loop(i,1,n) Wp->draw_segment(p[i],p[b[i]],green);
}
#endif
```

F.9 Trace.h

```
#ifndef TRACE_H
#define TRACE_H

#ifdef GRAPHICS
#include <LEDA/window.h>
extern window* Wp;
const int waitingtime=0;
#endif

extern int trace;

#endif
```

F.10 UniformGenerator.h

```
#ifndef UNIFORMGENERATOR_H
#define UNIFORMGENERATOR_H
```

```
// A uniform random number generator using the GNU g++ library
// random number generators.

#include <ACG.h>
#include <Uniform.h>

class UniformGenerator{
private:
  ACG* pRNG;
  Uniform* pRnd;
public:
  UniformGenerator(){};
  UniformGenerator(int seed=1){
    pRNG = new ACG(seed, 30);
    pRnd = new Uniform(0, 1, pRNG);
  }
  ~UniformGenerator(){
    delete pRNG;
    delete pRnd;
  }
  double rand(){
    return (*pRnd)();
  }
  double operator()(){
    return rand();
  }
};
typedef UniformGenerator *pUniformGenerator;

#endif
```

Appendix G

Travelling Salesman Test Problems

The problem data and optimal solutions are from TSPLIB[32], a library of travelling salesman problems, which is available by anonymous ftp from softlib.cs.rice.edu.For each problem the following are shown:

- The length of the optimal tour;
- A table showing the coordinates and optimal tour. For example, in problem EIL051 "city" 1 is at position (37,52) and its successor in the optimal tour is city 22;
- A picture of the optimal tour.

G.1 Test Problem EIL051

Optimal Tour Length 428.87

Points and Optimal Tour

1	(37, 52)	22	27	(30, 48)	51
2	(49, 49)	16	28	(43, 67)	3
3	(52, 64)	36	29	(58, 48)	2
4	(20, 26)	17	30	(58, 27)	9
5	(40, 30)	38	31	(37, 69)	28
6	(21, 47)	27	32	(38, 46)	1
7	(17, 63)	23	33	(46, 10)	45
8	(31, 62)	26	34	(61, 33)	30
9	(52, 33)	49	35	(62, 63)	20
10	(51, 21)	39	36	(63, 69)	35
11	(42, 41)	32	37	(32, 22)	5
12	(31, 32)	47	38	(45, 35)	11
13	(5, 25)	25	39	(59, 15)	33
14	(12, 42)	24	40	(5, 6)	41
15	(36, 16)	44	41	(10, 17)	13
16	(52, 41)	50	42	(21, 10)	19
17	(27, 23)	37	43	(5, 64)	7
18	(17, 33)	4	44	(30, 15)	42
19	(13, 13)	40	45	(39, 10)	15
20	(57, 58)	29	46	(32, 39)	12
21	(62, 42)	34	47	(25, 32)	18
22	(42, 57)	8	48	(25, 55)	6
23	(16, 57)	48	49	(48, 28)	10
24	(8, 52)	43	50	(56, 37)	21
25	(7, 38)	14	51	(30, 40)	46
26	(27, 68)	31			

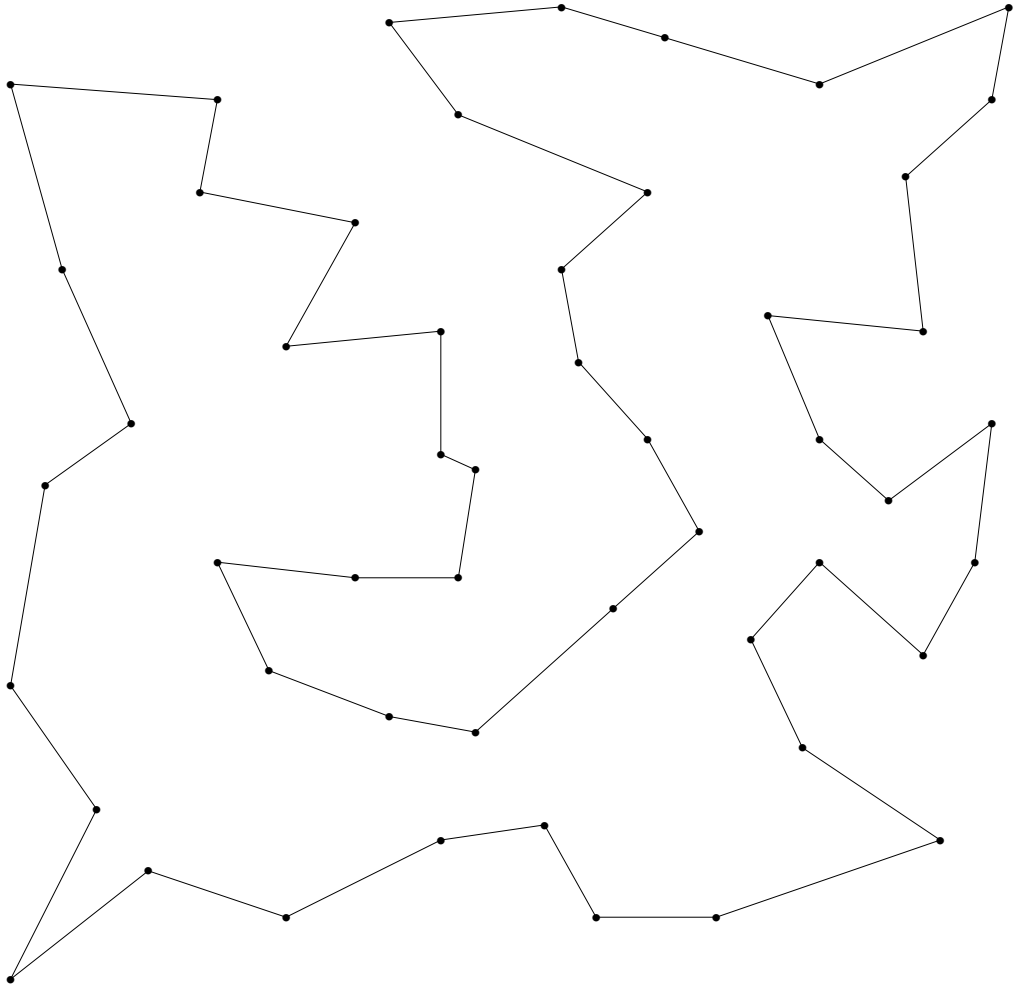


Figure G.1. Test Problem EIL051 optimal tour.

G.2 Test Problem EIL076

Optimal Tour Length 544.37

Points and Optimal Tour

1	(22, 22)	33	39	(30, 60)	72
2	(36, 26)	74	40	(30, 50)	17
3	(21, 45)	44	41	(12, 17)	43
4	(45, 35)	75	42	(15, 14)	64
5	(55, 20)	15	43	(16, 19)	42
6	(33, 34)	68	44	(21, 48)	32
7	(50, 50)	53	45	(50, 30)	29
8	(55, 45)	35	46	(51, 42)	52
9	(26, 59)	39	47	(50, 15)	36
10	(40, 66)	31	48	(48, 21)	30
11	(55, 65)	66	49	(12, 38)	23
12	(35, 51)	40	50	(15, 56)	18
13	(62, 35)	54	51	(29, 39)	6
14	(62, 57)	59	52	(54, 38)	27
15	(62, 24)	57	53	(55, 57)	14
16	(21, 36)	3	54	(67, 41)	19
17	(33, 44)	51	55	(10, 70)	25
18	(9, 56)	24	56	(6, 25)	41
19	(62, 48)	8	57	(65, 27)	13
20	(66, 14)	37	58	(40, 60)	12
21	(44, 13)	47	59	(70, 64)	11
22	(26, 13)	62	60	(64, 4)	70
23	(11, 28)	56	61	(36, 6)	21
24	(7, 43)	49	62	(30, 20)	73
25	(17, 64)	50	63	(20, 30)	16
26	(41, 46)	67	64	(15, 5)	22
27	(55, 34)	45	65	(50, 70)	38
28	(35, 16)	61	66	(57, 72)	65
29	(52, 26)	48	67	(45, 42)	34
30	(43, 26)	2	68	(38, 33)	4
31	(31, 76)	55	69	(50, 4)	71
32	(22, 53)	9	70	(66, 8)	20
33	(26, 29)	63	71	(59, 5)	60
34	(50, 40)	46	72	(35, 60)	58
35	(55, 50)	7	73	(27, 24)	1
36	(54, 10)	69	74	(40, 20)	28
37	(60, 15)	5	75	(40, 37)	76
38	(47, 66)	10	76	(40, 40)	26

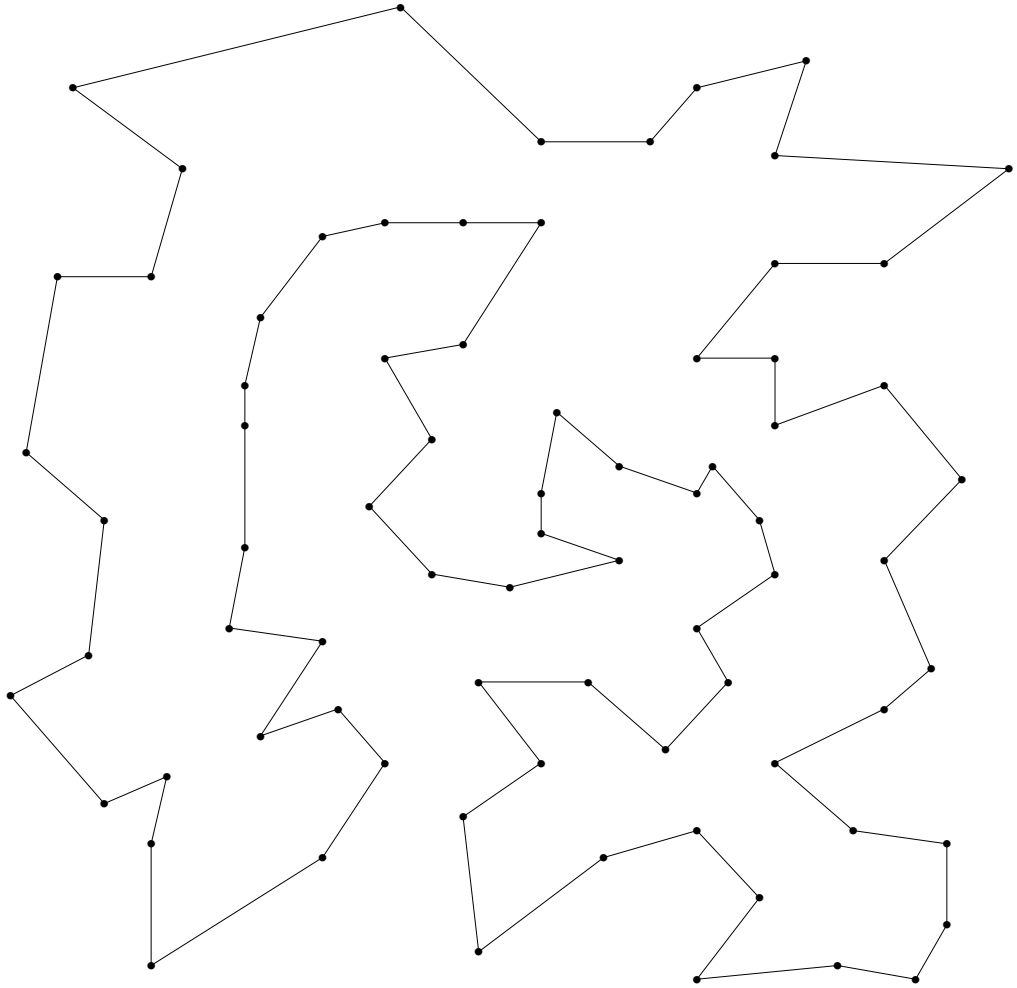


Figure G.2. Test Problem EIL076 optimal tour.

G.3 Test Problem KRO124

Optimal Tour Length 21285.44

Points and Optimal Tour

1	(1380, 939)	47	51	(2482, 1183)	87
2	(2848, 96)	44	52	(3854, 923)	78
3	(3510, 1671)	43	53	(376, 825)	88
4	(457, 334)	97	54	(2519, 135)	2
5	(3888, 666)	52	55	(2945, 1622)	7
6	(984, 965)	63	56	(953, 268)	80
7	(2721, 1482)	9	57	(2628, 1479)	20
8	(1286, 525)	92	58	(2097, 981)	61
9	(2716, 1432)	57	59	(890, 1846)	74
10	(738, 1325)	84	60	(2139, 1806)	77
11	(1251, 1832)	15	61	(2421, 1007)	51
12	(2728, 1698)	27	62	(2290, 1810)	60
13	(3815, 169)	76	63	(1115, 1052)	1
14	(3683, 1533)	3	64	(2588, 302)	40
15	(1247, 1945)	17	65	(327, 265)	4
16	(123, 862)	94	66	(241, 341)	26
17	(1234, 1946)	59	67	(1917, 687)	58
18	(252, 1240)	79	68	(2991, 792)	85
19	(611, 673)	90	69	(2573, 599)	64
20	(2576, 1676)	12	70	(19, 674)	66
21	(928, 1700)	72	71	(3911, 1673)	14
22	(53, 857)	70	72	(872, 1559)	10
23	(1807, 1711)	98	73	(2863, 558)	68
24	(274, 1420)	18	74	(929, 1766)	21
25	(2574, 946)	81	75	(839, 620)	19
26	(178, 24)	65	76	(3893, 102)	33
27	(2678, 1825)	86	77	(2178, 1619)	23
28	(1795, 962)	67	78	(3822, 899)	96
29	(3384, 1498)	34	79	(378, 1048)	53
30	(3520, 1079)	48	80	(1178, 100)	31
31	(1256, 61)	89	81	(2599, 901)	69
32	(1424, 1728)	11	82	(3416, 143)	95
33	(3913, 192)	37	83	(2961, 1605)	55
34	(3085, 1528)	83	84	(611, 1384)	36
35	(2573, 1969)	62	85	(3113, 885)	82
36	(463, 1670)	99	86	(2597, 1830)	35
37	(3875, 598)	5	87	(2586, 1286)	25
38	(298, 1513)	24	88	(161, 906)	16
39	(3479, 821)	30	89	(1429, 134)	42
40	(2542, 236)	54	90	(742, 1025)	49
41	(3955, 1743)	71	91	(1625, 1651)	45
42	(1323, 280)	8	92	(1187, 706)	75
43	(3447, 1830)	46	93	(1787, 1009)	28
44	(2936, 337)	50	94	(22, 987)	22
45	(1621, 1830)	32	95	(3640, 43)	13
46	(3373, 1646)	29	96	(3756, 882)	39
47	(1393, 1368)	93	97	(776, 392)	56
48	(3874, 1318)	100	98	(1724, 1642)	91
49	(938, 955)	6	99	(198, 1810)	38
50	(3022, 474)	73	100	(3950, 1558)	41

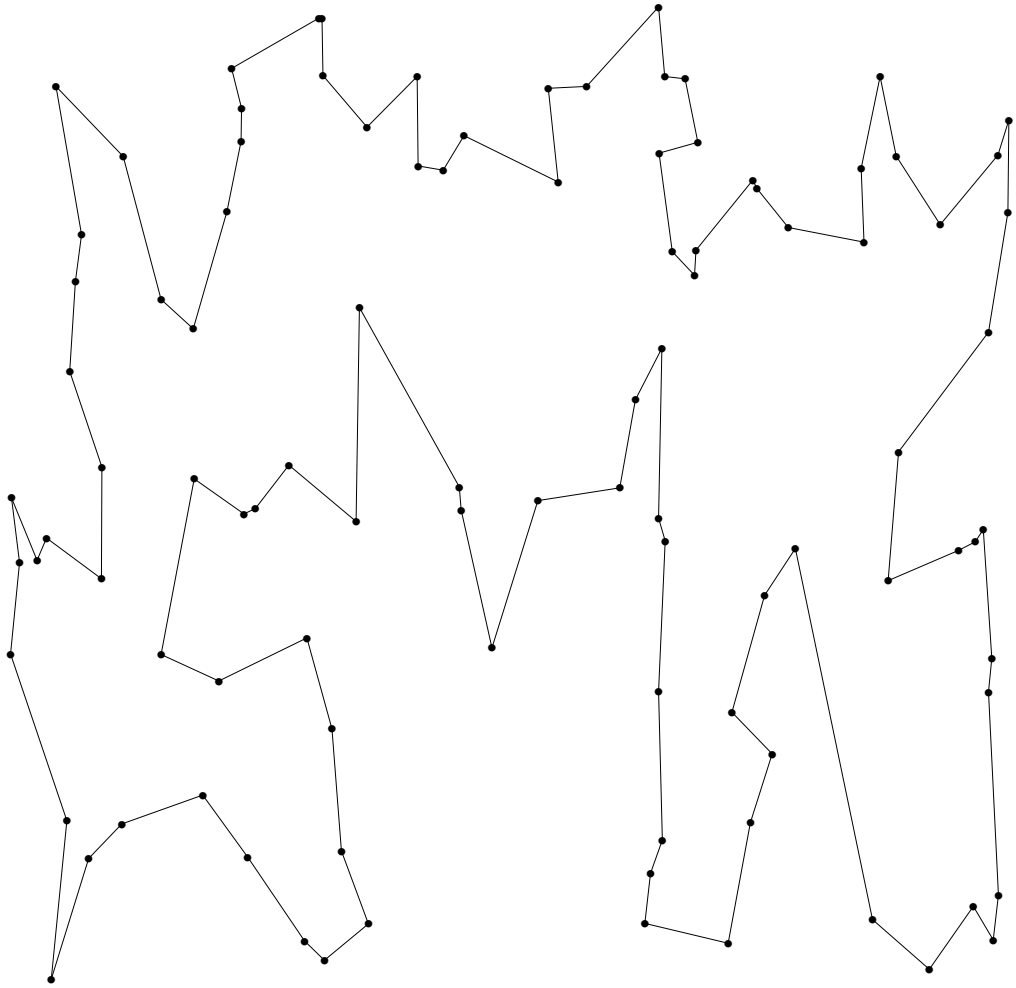


Figure G.3. Test Problem KRO124 optimal tour.

G.4 Test Problem KRO126

Optimal Tour Length 20750.76

Points and Optimal Tour

1	(1357, 1905)	85	51	(86, 1065)	16
2	(2650, 802)	54	52	(14, 454)	11
3	(1774, 107)	69	53	(1327, 1893)	1
4	(1307, 964)	93	54	(2773, 1286)	6
5	(3806, 746)	86	55	(2469, 1838)	67
6	(2687, 1353)	75	56	(3835, 963)	43
7	(43, 1957)	26	57	(1031, 428)	74
8	(3092, 1668)	17	58	(3853, 1712)	98
9	(185, 1542)	78	59	(1868, 197)	73
10	(834, 629)	14	60	(1544, 863)	4
11	(40, 462)	84	61	(457, 1607)	32
12	(1183, 1391)	40	62	(3174, 1064)	50
13	(2048, 1628)	79	63	(192, 1004)	51
14	(1097, 643)	36	64	(2318, 1925)	20
15	(1838, 1732)	13	65	(2232, 1374)	80
16	(234, 1118)	37	66	(396, 828)	44
17	(3314, 1881)	25	67	(2365, 1649)	47
18	(737, 1285)	49	68	(2499, 658)	35
19	(779, 777)	92	69	(1410, 307)	60
20	(2312, 1949)	42	70	(2990, 214)	23
21	(2576, 189)	89	71	(3646, 1018)	56
22	(3078, 1541)	8	72	(3394, 1028)	83
23	(2781, 478)	21	73	(1779, 90)	3
24	(705, 1812)	46	74	(1058, 372)	100
25	(3409, 1917)	90	75	(2933, 1459)	22
26	(323, 1714)	61	76	(3099, 173)	70
27	(1660, 1556)	15	77	(2178, 978)	30
28	(3729, 1188)	39	78	(138, 1610)	82
29	(693, 1383)	18	79	(2082, 1753)	64
30	(2361, 640)	68	80	(2302, 1127)	77
31	(2433, 1538)	65	81	(805, 272)	97
32	(554, 1825)	24	82	(22, 1617)	7
33	(913, 317)	45	83	(3213, 1085)	62
34	(3586, 1909)	58	84	(99, 536)	48
35	(2636, 727)	2	85	(1533, 1780)	27
36	(1000, 457)	57	86	(3564, 676)	72
37	(482, 1337)	9	87	(29, 6)	52
38	(3704, 1082)	71	88	(3808, 1375)	28
39	(3635, 1174)	38	89	(2221, 291)	41
40	(1362, 1526)	53	90	(3499, 1885)	34
41	(2049, 417)	59	91	(3124, 408)	76
42	(2552, 1909)	55	92	(781, 671)	10
43	(3939, 640)	5	93	(1027, 1041)	99
44	(219, 898)	63	94	(3249, 378)	91
45	(812, 351)	81	95	(3297, 491)	94
46	(901, 1552)	29	96	(213, 220)	87
47	(2513, 1572)	31	97	(721, 186)	96
48	(242, 584)	66	98	(3736, 1542)	88
49	(826, 1226)	12	99	(868, 731)	19
50	(3278, 799)	95	100	(960, 303)	33

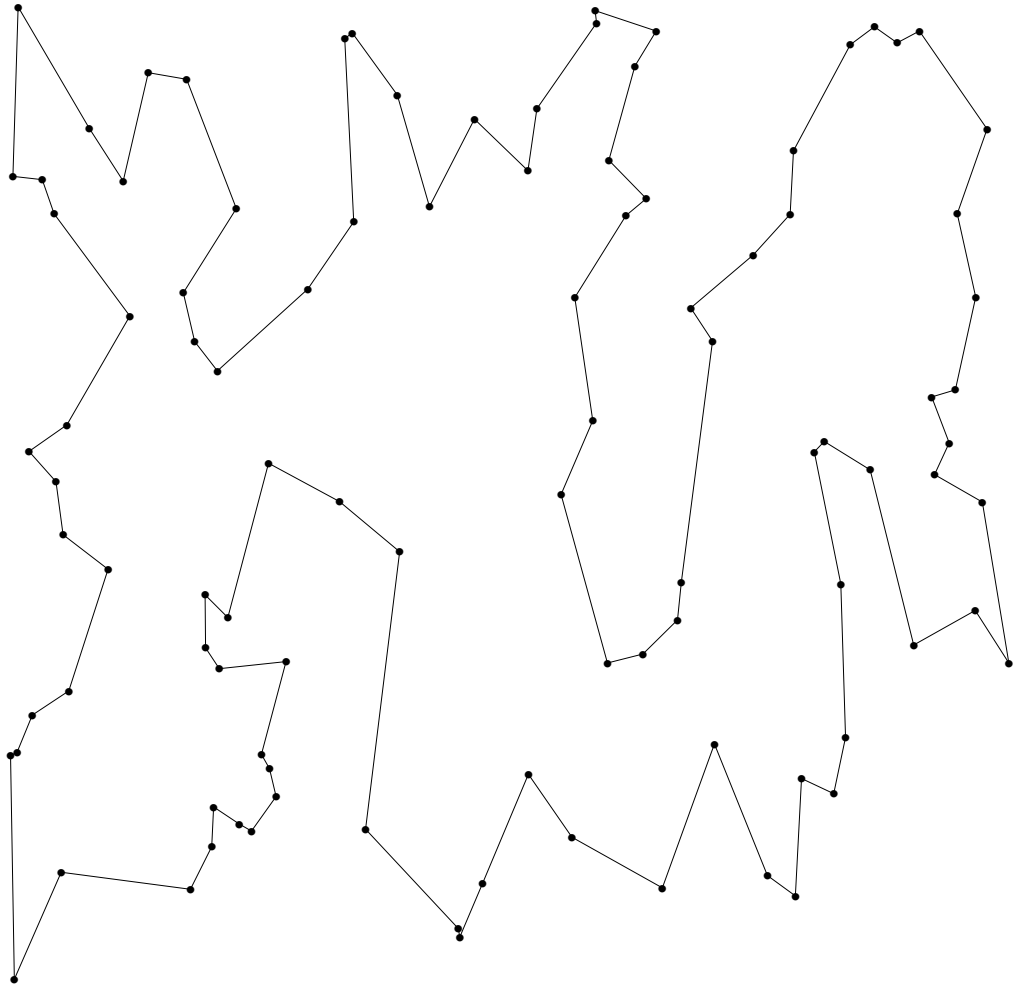


Figure G.4. Test Problem KRO126 optimal tour.

G.5 Test Problem KRO127

Optimal Tour Length 21294.29

Points and Optimal Tour

1	(2995, 264)	50	51	(387, 199)	19
2	(202, 233)	100	52	(2901, 920)	46
3	(981, 848)	83	53	(931, 512)	60
4	(1346, 408)	71	54	(1766, 692)	91
5	(781, 670)	53	55	(401, 980)	92
6	(1009, 1001)	18	56	(149, 1629)	95
7	(2927, 1777)	74	57	(2214, 1977)	31
8	(2982, 949)	52	58	(3805, 1619)	29
9	(555, 1121)	10	59	(1179, 969)	3
10	(464, 1302)	21	60	(1017, 333)	44
11	(3452, 637)	90	61	(2834, 1512)	76
12	(571, 1982)	28	62	(634, 294)	5
13	(2656, 128)	1	63	(1819, 814)	54
14	(1623, 1723)	65	64	(1393, 859)	85
15	(2067, 694)	69	65	(1768, 1578)	86
16	(1725, 927)	63	66	(3023, 871)	8
17	(3600, 459)	94	67	(3248, 1906)	75
18	(1109, 1196)	82	68	(1632, 1742)	14
19	(366, 339)	37	69	(2223, 990)	84
20	(778, 1282)	9	70	(3868, 697)	22
21	(386, 1616)	47	71	(1541, 354)	39
22	(3918, 1217)	58	72	(2374, 1944)	57
23	(3332, 1049)	42	73	(1962, 389)	15
24	(2597, 349)	27	74	(3007, 1524)	61
25	(811, 1295)	20	75	(3220, 1945)	7
26	(241, 1069)	87	76	(2356, 1568)	80
27	(2658, 360)	88	77	(1604, 706)	43
28	(394, 1944)	56	78	(2028, 1736)	99
29	(3786, 1862)	36	79	(2581, 121)	13
30	(264, 36)	35	80	(2221, 1578)	72
31	(2050, 1833)	78	81	(2944, 632)	38
32	(3538, 125)	97	82	(1082, 1561)	25
33	(1646, 1817)	68	83	(997, 942)	40
34	(2993, 624)	81	84	(2334, 523)	41
35	(547, 25)	51	85	(1264, 1090)	59
36	(3373, 1902)	67	86	(1699, 1294)	96
37	(460, 267)	62	87	(235, 1059)	55
38	(3060, 781)	66	88	(2592, 248)	79
39	(1828, 456)	73	89	(3642, 699)	11
40	(1021, 962)	6	90	(3599, 514)	17
41	(2347, 388)	24	91	(1766, 678)	77
42	(3535, 1112)	89	92	(240, 619)	48
43	(1529, 581)	64	93	(1272, 246)	4
44	(1203, 385)	93	94	(3503, 301)	32
45	(1787, 1902)	33	95	(80, 1533)	26
46	(2740, 1101)	23	96	(1677, 1238)	16
47	(555, 1753)	12	97	(3766, 154)	98
48	(47, 363)	2	98	(3946, 459)	49
49	(3935, 540)	70	99	(1994, 1852)	45
50	(3062, 329)	34	100	(278, 165)	30

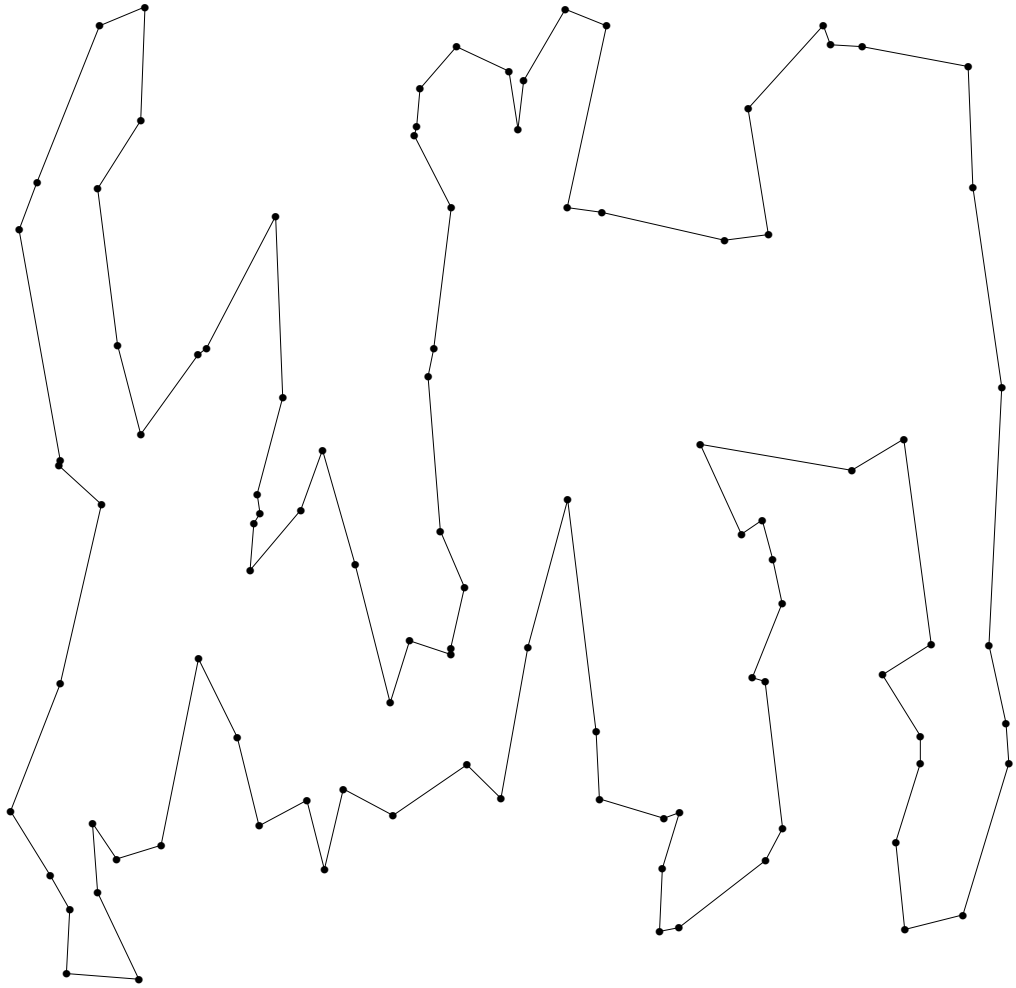


Figure G.5. Test Problem KRO127 optimal tour.

G.6 Test Problem LIN105

Optimal Tour Length 14383.00

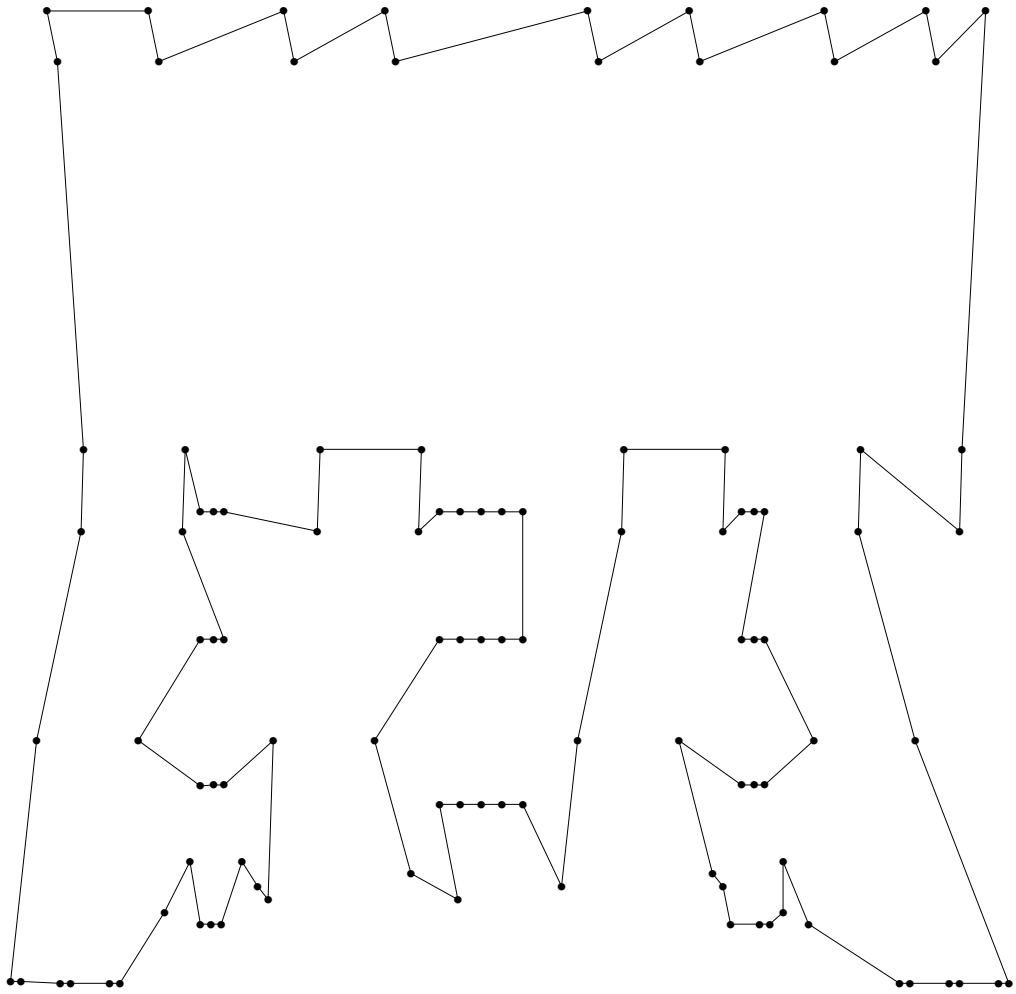


Figure G.6. Test Problem LIN105 optimal tour.

Points and Optimal Tour

1	(63, 71)	2	54	(1551, 496)	51
2	(94, 71)	6	55	(1551, 291)	56
3	(142, 370)	1	56	(1614, 291)	59
4	(173, 1276)	5	57	(1614, 496)	54
5	(205, 1213)	9	58	(1614, 654)	57
6	(213, 69)	7	59	(1732, 189)	105
7	(244, 69)	10	60	(1811, 1276)	39
8	(276, 630)	3	61	(1843, 1213)	60
9	(283, 732)	8	62	(1913, 630)	63
10	(362, 69)	11	63	(1921, 732)	70
11	(394, 69)	15	64	(2087, 370)	67
12	(449, 370)	19	65	(2118, 1276)	61
13	(480, 1276)	4	66	(2150, 1213)	65
14	(512, 1213)	13	67	(2189, 205)	68
15	(528, 157)	103	68	(2220, 189)	71
16	(583, 630)	17	69	(2220, 630)	74
17	(591, 732)	18	70	(2228, 732)	69
18	(638, 654)	25	71	(2244, 142)	78
19	(638, 496)	24	72	(2276, 315)	64
20	(638, 314)	12	73	(2276, 496)	76
21	(638, 142)	22	74	(2276, 654)	75
22	(669, 142)	29	75	(2315, 654)	81
23	(677, 315)	20	76	(2315, 496)	80
24	(677, 496)	27	77	(2315, 315)	72
25	(677, 654)	26	78	(2331, 142)	82
26	(709, 654)	36	79	(2346, 315)	77
27	(709, 496)	16	80	(2346, 496)	86
28	(709, 315)	23	81	(2346, 654)	73
29	(701, 142)	30	82	(2362, 142)	83
30	(764, 220)	31	83	(2402, 157)	84
31	(811, 189)	32	84	(2402, 220)	85
32	(843, 173)	33	85	(2480, 142)	91
33	(858, 370)	28	86	(2496, 370)	79
34	(890, 1276)	14	87	(2528, 1276)	66
35	(921, 1213)	34	88	(2559, 1213)	87
36	(992, 630)	37	89	(2630, 630)	90
37	(1000, 732)	42	90	(2638, 732)	98
38	(1197, 1276)	35	91	(2756, 69)	92
39	(1228, 1213)	38	92	(2787, 69)	96
40	(1276, 205)	49	93	(2803, 370)	89
41	(1299, 630)	43	94	(2835, 1276)	88
42	(1307, 732)	41	95	(2866, 1213)	94
43	(1362, 654)	46	96	(2906, 69)	97
44	(1362, 496)	104	97	(2937, 69)	101
45	(1362, 291)	48	98	(2937, 630)	99
46	(1425, 654)	52	99	(2945, 732)	100
47	(1425, 496)	44	100	(3016, 1276)	95
48	(1425, 291)	50	101	(3055, 69)	102
49	(1417, 173)	45	102	(3087, 69)	93
50	(1488, 291)	55	103	(606, 220)	21
51	(1488, 496)	47	104	(1165, 370)	40
52	(1488, 654)	53	105	(1780, 370)	62
53	(1551, 654)	58			

Appendix H

Listing of the Euclidean Steiner Tree Problem Simulated Annealing Program

This appendix is a listing of the C++ program for finding the solution of a Euclidean Steiner tree problem using simulated annealing. A polynomial cooling schedule is used. The file containing **main()** is Main.cc.[1]

The following program files are not listed in this appendix. They are identical to those in the listing of the travelling salesman simulated annealing code. The files are: Annealer.h, Annealer.cc, Generator.h, Problem.h, Trace.h and UniformGenerator.h.

The most important component of the Steiner simulated annealing code is the Delaunay triangulation object. This is from the LEDA library. It is useful because because it is a dynamic implementation. That is, it is possible to add and delete points at will and always have a Delaunay triangulation on hand without having to start from scratch each time to determine the triangulation. The implementation is complex and is not discussed here. The triangulation is used because it provides an easy way of finding the minimum spanning tree of a set of points. The Delaunay triangulation is discussed in Section 4.2.1. Other code for finding Delaunay triangulations was experimented with. However, this was chosen because it was part of the familiar LEDA library and appeared to be faster. Unfortunately the code is not without bugs. Occasionally it will crash. Annoying when it happens but thankfully not too frequent.

The LEDA Delaunay triangulation code was slightly modified to provide necessary output of a minimum spanning tree as a series of edges. The additions to the LEDA code are shown in the listings of files DT.h.diff and DT.cc.diff. These files contain the output of the Unix `diff` command used to compare the LEDA files delaunay_tree.h and _delaunay_tree.c with DT.h and DT.cc respectively.

The inputs, text outputs and graphical output are are of the same to those for the travelling salesman annealing program. Tables F.1, F.2 and F.3 describe

[1]The preamble to the program listing in Appendix A describes the software and hardware used for all programs.

the inputs and outputs of the travelling salesman program.

H.1 Makefile

```
# $Id: Makefile,v 2.3 1994/06/23 00:57:53 geoff Exp $
# $Log: Makefile,v $
# Revision 2.3 1994/06/23 00:57:53 geoff
# Added $Log$
#
#This Makefile takes the .cc files defined by SOURCES and compiles and links
#them to give the program with name defined by PROGRAM.

LEDA = /u/grads/geoff/LEDA

#LEDA is a handy library of graph, list, array and geometrical objects.

X11 = /usr/openwin/lib

TARGET = ESTP-2
SOURCES = Main Annealer ESTPProblem DT

OBJECTS = $(SOURCES:%=%.o)

#OBJECTS is a list of .o filenames corresponding to SOURCES.

.SILENT:

#To see all that is happening comment out the .SILENT line. The horrific
#commands that do the compiling and linking will be shown.

.SUFFIXES: .cc .o
.KEEP_STATE:

CCC = g++

#g++ is the GNU C++ compiler and linker.

INCLUDES= -I$(LEDA)/incl

#INCLUDES is the directory in which to search for LEDA .h files.

CCFLAGS = -DTRACE -O2

#-O2 -DBEST -DGRAPHICS -DTRACE

#-O2 is the optimisation flag for the g++ compiler and linker.

CPPFLAGS=
LDFLAGS =
LEDALIBS= -L$(LEDA) -lP -lG -lL -lWx

#LEDALIBS is the directory and list of library archive files in which to
#for LEDA objects and programs. -lP means libP.a etc.

X11LIBS = -L$(X11) -lxview -lolgx -lX11
```

```
COMPILE = $(CCC) $(CCFLAGS) $(INCLUDES) $(CPPFLAGS)-c

#COMPILE is the compile command without linking.

LINK = $(CCC) $(CCFLAGS) $(CPPFLAGS) $(LDFLAGS)

#LINK is the link command

.cc.o:
        @echo Compiling -- $*
        $(COMPILE) $<

#.cc.o means for each .cc with an old .o or no .o perform the compile command.

$(TARGET): $(OBJECTS)
          @echo Linking --- $@
          $(LINK)-o $(TARGET) $(OBJECTS) $(LEDALIBS) $(X11LIBS) -lm

#(TARGET): $(OBJECTS) means link any new .o files to give a new PROGRAM file
#or link all .o files to give a PROGRAM file if no PROGRAM file exists.
```

H.2 DT.h.diff

```
1d0
< // Id: DT.h,v 2.11 1994/06/23 00:01:33 geoffExp
72,79d70
< #include <LEDA/prio.h>
< #include <LEDA/array.h>
< #include <LEDA/list.h>
< #include "Point.h"
< #include "link.h"
< typedef array<pPoint> APType;
< typedef APType *pAPType;
< typedef list<link> MSTType;
205,206c197
< void add_link(noeud, priority_queue<DTlink,double>&);
< void links(priority_queue<DTlink,double>&);
---
>
209d199
< int size();
213a204,205
>
>
226,229d217
< void change_inf(DT_item p, itype i){
< delaunay_tree::change_inf(p,Convert(i));
< }
<
234,235d221
<
< typedef DELAUNAY_TREE<int> DTType;
```

H.3 DT.cc.diff

```
31c30
< #include "DT.h"
---
> #include <LEDA/impl/delaunay_tree.h>
1556,1598d1554
< void delaunay_tree::add_link(noeud n, priority_queue<DTlink,double>& PQ){
<
< noeud* stack = new noeud[counter+10];
< int top = 0;
< stack[0] = n;
< n→stacked = flag;
< int both=0;
<
< int i;
<
< while (top ≥ 0){
< n = stack[top--];
<
< n→visite = flag;
<
< for (i=U; i≤W; i++)
< if ( both || n→voisin[i]→visite ≠ flag ){
< int j = (i==U) ? V : U ;
< int k = U + V + W -i -j ;
< if (( n→type == Fini ) || ((i == W) && (n→type==Infini1) ) ){
< DTlink dtl(n→s[j], n→s[k]);
< double ax=n→s[j]→x;
< double ay=n→s[j]→y;
< double bx=n→s[k]→x;
< double by=n→s[k]→y;
< point pa(ax,ay);
< point pb(bx,by);
< double l=pa.distance(pb);
< PQ.insert(dtl,l);
< }
< }
<
< for (i=W; i≥U; i--){
< noeud v = n→voisin[i];
< if ( v→stacked ≠ flag ){
< stack[++top] = v;
< v→stacked = flag;
< }
< }
<
1600,1602d1555
<
< delete stack;
<
1605,1615d1557
< void delaunay_tree::links(priority_queue<DTlink,double>& PQ){
<
< if (arbre == nouveau) return;
< flag++;
```

```
< add_link(nouveau,PQ);
<
< }
<
< int delaunay_tree::size(){
< return counter;
< }
```

H.4 ESTPProblem.h

```
// Id : ESTPProblem.h, v2.31994/08/2307 : 56 : 08geoffExp
#ifndef ESTPPROBLEM_H
#define ESTPPROBLEM_H

// The Euclidean Steiner Tree Problem class

#include <LEDA/array.h>
#include <LEDA/plane.h>
#include <LEDA/list.h>
#include <LEDA/set.h>
#include <LEDA/window.h>

#include "Problem.h"
#include "UniformGenerator.h"
#include "DT.h"

typedef set<point> ESTPSolution;
typedef ESTPSolution *pESTPSolution;

typedef array<point> Points;
typedef Points *pPoints;

enum possibleChanges {ADD, DELETE, REPLACE, NONE};

class ESTPProblem : public Problem{

public:
  ESTPProblem();
  ~ESTPProblem();
  void generateNeighbour();
  void updateSolution();
  int getL();
  void printSolution();
  void printNeighbour();
  void undoChanges();

#ifdef GRAPHICS
  void plotSolution(color c=black);
  void plotPoints();
  void plotChange();
#endif

protected:
  // problem size
  int n;
```

```
  // number of Steiner points in solution
  int k;
  // array of point coordinates
  pPoints positions;
  // current configuration
  pESTPSolution solution;
  // neighbour configuration
  pESTPSolution neighbour;
  // random number generator
  pUniformGenerator URNG;
  // Delaunay triangulation
  DTType DT;
  // minimum spanning tree
  MSTType MST;
  // array of pointers to objects containing point coordinates
  // and pointer to the point's item in the Delaunay triangulation
  pAPType AP;
  // new and old items in the Delaunay triangulation
  DT_item newdt;
  DT_item olddt;
  // new and old elements in the AP array
  pPoint oldpt;
  pPoint newpt;
  // numerous objects for doing geometry
  int x,y,z;
  point Px,Py,Pz,Ps,Exy;
  segment Sxy,Axis;
  circle Cxy;
  possibleChanges changeType;
  double mstlength;
  // cost function
  double distance(ESTPSolution& s);
  void newNeighbour();
  void traceESTPProblem();
  void traceGenerateNeighbour();
  void addPoint();
  void deletePoint();
  void replacePoint();
  bool Opposite(const point& x, const point& y, line& l);
  void traceAddPoint();
  void traceDeletePoint();
  void traceReplacePoint();
};

typedef ESTPProblem *pESTPProblem;

#endif
```

H.5 ESTPProblem.cc

```
// Id : ESTPProblem.cc, v2.41994/08/2307 : 56 : 08geoffExp
#include "ESTPProblem.h"

#include <LEDA/partition.h>
```

```
#include "Trace.h"

// constructor
ESTPProblem::ESTPProblem(){
  // read the number of points
  n=read_int();
  // read the random nmber generator seed
  int seed=read_int();
  URNG=new UniformGenerator(seed);
  positions=new Points(1,n);
  solution=new ESTPSolution;
  neighbour=new ESTPSolution;
  AP=new APType(1,2*n-2);
  ESTPSolution &s=*solution;
  Points &p=*positions;
  APType &a=*AP;
  // read the points
  p.read();
  register int i;
  // construct the initial Delaunay triangulation
  loop(i,1,n){
    newdt=DT.insert(p[i],i);
    newpt=new Point(p[i],newdt);
    a[i]=newpt;
  }
  s.clear();
  k=0;
  // get the current solution cost
  mstlength=100;
  solutionDistance=distance(s);
  mstlength=solutionDistance;
  traceESTPProblem();
}

// destructor
ESTPProblem::~ESTPProblem(){
  delete solution;
  delete neighbour;
  delete URNG;
  delete AP;
  delete positions;
}

// perform a transition
void ESTPProblem::generateNeighbour(){
  changeType=NONE;
  // one third change of doing an add, delete or replace
  // although the value of k may change that
  double PrAdd=1.0/3.0;
  double PrDelete=2.0/3.0;
  double u;
  // repeat until a change is found
  do{
    u=URNG->rand();
    if ((u<PrAdd)&&(k<(n-2))){
      addPoint();
    }
```

```
    else{
      if ((u<PrDelete)&&(k>0)){
        deletePoint();
      }
      else{
        if (k>0){
          replacePoint();
        }
      }
    }
  } while (changeType==NONE);
  // get the cost of the neighbour
  neighbourDistance=distance(*neighbour);
  change=neighbourDistance-solutionDistance;
#ifdef TRACE
  traceGenerateNeighbour();
#endif
}

// the add transition
void ESTPProblem::addPoint(){
  do{
    // find three distinct points with which to construct a new point
    x=int(URNG->rand()*(n+k))+1;
    while (x==(y=int(URNG->rand()*(n+k))+1));
    z=int(URNG->rand()*(n+k))+1;
    while ((x==z)||(y==z)) z=int(URNG->rand()*(n+k))+1;
    //    cout << string("%3d%3d%3d\n", x, y, z);
    APType &a=*AP;
    Px=a[x]->pt;
    Py=a[y]->pt;
    Pz=a[z]->pt;
    // find the equilateral point and axis
    Exy=Py.rotate(Px,PI/3);
    if (!Opposite(Exy,Pz,line(Px,Py))) Exy=Px.rotate(Py,PI/3);
    Sxy=segment(Px,Py);
    Axis=segment(Pz,Exy);
    point p;
    // the axis must cross the E-arc
    if (Sxy.intersection(Axis,p)){
      //    cout << "Axis intersects Segment xy\n";
      // find the equilateral circle
      double Cx=(Px.xcoord()+Py.xcoord()+Exy.xcoord())/3;
      double Cy=(Px.ycoord()+Py.ycoord()+Exy.ycoord())/3;
      point C(Cx, Cy);
      Cxy=circle(C, C.distance(Px));
      // the third point must be outside the E-circle
      if (Cxy.outside(Pz)){
        //        cout << "z is outside E-circle\n";
        // find the intersection of the E-circle and the axis
        list<point> plist;
        plist=Cxy.intersection(line(Axis));
        Ps=plist.pop();
        double d=Exy.distance(Ps);
        forall(p,plist)
          if (Exy.distance(Ps)>d)
            Ps=p;
```

```
        ESTPSolution &s=*solution;
        // the new point must not already be in the solution
        if (!s.member(Ps)){
          //          cout << "Steiner point is not in current solution ";
          //          cout << Ps; newline;
          //          forall(p,s){
          //            cout << p; newline;
          //          }
          // add it to the triangulation
          ++k;
          newdt=DT.insert(Ps,n+k);
          //          cout << "Steiner point added to DT\n";
          newpt=new Point(Ps,newdt);
          //          traceAddPoint();

          //          list<DT_item> listdt;
          //          listdt.clear();
          //          DT.all_items(listdt);
          //          DT_item dt;
          //          forall(dt,listdt){
          //            cout << DT.inf(dt) << " " << DT.key(dt); newline;
          //          }

          a[n+k]=newpt;
          s.insert(Ps);
          changeType=ADD;
#ifdef TRACE
          traceAddPoint();
#endif
        }
      }
    }
  } while (changeType≠ADD);
}

// the delete transition
void ESTPProblem::deletePoint(){
  // find a random point and remove it
  x=int(URNG→rand()*k)+n+1;
  APType &a=*AP;
  oldpt=a[x];
  ESTPSolution &s=*solution;
  s.del(oldpt→pt);
  olddt=oldpt→dt;
  DT.del_item(olddt);
  register int i;
  loop(i,x+1,n+k){
    DT.change_inf(a[i]→dt,i-1);
    a[i-1]=a[i];
  }

  //   traceDeletePoint();
  //   list<DT_item> listdt;
  //   listdt.clear();
  //   DT.all_items(listdt);
  //   DT_item dt;
  //   forall(dt,listdt){
```

```
  //     cout << DT.inf(dt) << " " << DT.key(dt); newline;
  //   }

  --k;
  changeType=DELETE;
#ifdef TRACE
  traceDeletePoint();
#endif
}

// the replace transition
void ESTPProblem::replacePoint(){
  int oldk=k;
#ifdef TRACE
  traceReplacePoint();
#endif
  deletePoint();
  while (k≠oldk) addPoint();
  changeType=REPLACE;
}

// it is necessary to be able to undo the changes to the triangulation
// if a transition is rejected
void ESTPProblem::undoChanges(){
  //cout << "UNDO\n";
  APType &a=∗AP;
  ESTPSolution &s=∗solution;
  if ((changeType==REPLACE)||(changeType==ADD)){
    traceGenerateNeighbour();
    s.del(newpt→pt);
    DT.del_item(newpt→dt);
    register int i;
    delete newpt;
    --k;
  }
  if ((changeType==REPLACE)||(changeType==DELETE)){
    traceGenerateNeighbour();
    ++k;
    newdt=DT.insert(oldpt→pt,n+k);
    a[n+k]=new Point(oldpt→pt,newdt);
    s.insert(oldpt→pt);
  }

  //   list<DT_item> listdt;
  //   listdt.clear();
  //   DT.all_items(listdt);
  //   DT_item dt;
  //   forall(dt,listdt){
  //     cout << DT.inf(dt) << " " << DT.key(dt); newline;
  //   }

#ifdef GRAPHICS
  solutionDistance=distance(s);
#endif
}

// a small amout of housekeeping is necessary if a transition
```

```
// is accepted
void ESTPProblem::updateSolution(){
  //cout << "ACCEPT\n";
  solutionDistance=neighbourDistance;
  if ((changeType==DELETE)||(changeType==REPLACE))
    delete oldpt;

  //   list<DT_item> listdt;
  //   listdt.clear();
  //   DT.all_items(listdt);
  //   DT_item dt;
  //   forall(dt,listdt){
  //     cout << DT.inf(dt) << " " << DT.key(dt); newline;
  //   }
}

// the chain length
int ESTPProblem::getL(){
  //  return 3;
  return n*n;
  //  return (n-2)*(2*n-3)*(2*n-4)*(2*n-5)/6;
}

void ESTPProblem::printSolution(){
  ESTPSolution &s=*solution;
  point p;
  cout << string("\n%21d", k);
  forall(p,s)
    cout << string("\n%21.7f%15.7f", p.xcoord(), p.ycoord());
}

void ESTPProblem::printNeighbour(){
}

inline double ESTPProblem::distance(ESTPSolution& s){
  double t=0;
  priority_queue<DTlink,double> PQ;
  array<partition_item> P(1,n+k);
  partition PT;
  register int i;
  loop(i,1,n+k) P[i]=PT.make_block();
  DT.links(PQ);
#ifdef GRAPHICS
  MST.clear();
#endif
  i=0;
  while (i<(n+k-1)){
    pq_item pq=PQ.find_min();
    DTlink dt=PQ.key(pq);
    int s=DT.inf(dt.s);
    int f=DT.inf(dt.f);
    if (!PT.same_block(P[s],P[f])){
#ifdef GRAPHICS
      link lk(s,f);
      MST.append(lk);
#endif
      PT.union_blocks(P[s],P[f]);
```

```
      ++i;
      t+=PQ.inf(pq);
    }
    PQ.del_item(pq);
  }
  return t/mstlength*100.0;
}

inline void ESTPProblem::newNeighbour(){
};

void ESTPProblem::traceESTPProblem(){
  if (trace≥10){
    Points &p=*positions;
    int i;
    loop(i, 1, n) cout ≪ string("%6d%15.7f%15.7f\n",
                                i, p[i].xcoord(),p[i].ycoord());
#ifdef GRAPHICS
    MST.print('\n'); newline;
#endif
    cout ≪ string("%14.7f\n",solutionDistance);
  }
}

void ESTPProblem::traceGenerateNeighbour(){
  if (trace≥10){
    cout ≪ string("%21.7f", solutionDistance);
    cout ≪ string("%15.7f%15.7f ", neighbourDistance, change);
    newline;
  }
}

#ifdef GRAPHICS
void ESTPProblem::plotPoints(){
  Points &p=*positions;
  ESTPSolution &s=*solution;
  point q;
  register int i;
  loop(i,1,n) Wp→draw_filled_node(p[i]);
  forall(q,s) Wp→draw_filled_node(q,red);
}

void ESTPProblem::plotSolution(color c=black){
  plotPoints();
  APType &a=*AP;
  link l;
  forall(l,MST) Wp→draw_edge(a[l.s]→pt,a[l.f]→pt,c);
}

void ESTPProblem::plotChange(){
  if ((changeType==REPLACE)||(changeType==DELETE)){
    Wp→set_node_width(5);
    Wp→draw_filled_node(oldpt→pt,orange);
    Wp→set_node_width(3);
    if (changeType==REPLACE){
      Wp→draw_text(0,Wp→ymax()-0.1,
                   string("%8.4f Replace",solutionDistance));
```

```
    }
    else{
      Wp→draw_text(0,Wp→ymax()-0.1,
                   string("%8.4f Delete ",solutionDistance));
    }
  }
  if ((changeType==REPLACE)||(changeType==ADD)){
    Wp→set_node_width(5);
    Wp→draw_filled_node(newpt→pt,blue);
    Wp→draw_filled_node(Px,green);
    Wp→draw_filled_node(Py,green);
    Wp→draw_filled_node(Pz,green);
    Wp→draw_filled_node(Exy,green);
    Wp→draw_circle(Cxy,green);
    Wp→draw_edge(Exy,Pz,green);
    Wp→set_node_width(3);
    if (changeType==REPLACE){
      Wp→draw_text(0,Wp→ymax()-0.1,
                   string("%8.4f Replace",solutionDistance));
    }
    else{
      Wp→draw_text(0,Wp→ymax()-0.1,
                   string("%8.4f Add ",solutionDistance));
    }
  }
}
#endif

void ESTPProblem::traceAddPoint(){
  if (trace≥9){
    cout ≪ "ADD ---------------------------------\n";
    cout ≪ string("%4d ", x) ≪ Px; newline;
    cout ≪ string("%4d ", y) ≪ Py; newline;
    cout ≪ string("%4d ", z) ≪ Pz; newline;
    cout ≪ string("%4d ", n+k) ≪ newpt→pt; newline;
  }
}

void ESTPProblem::traceDeletePoint(){
  if (trace≥9){
    cout ≪ "DELETE ------------------------------\n";
    cout ≪ string("%4d ", x) ≪ oldpt→pt; newline;
  }
}

void ESTPProblem::traceReplacePoint(){
  if (trace≥9)
    cout ≪ "REPLACE -----------------------------\n";
}
bool ESTPProblem::Opposite(const point& x, const point& y, line& l){
  if (!l.vertical()){
    double lx = l.y_proj(x.xcoord());
    double ly = l.y_proj(y.xcoord());
    if ((lx<x.ycoord()) && ly<y.ycoord()) return false;
    if ((lx>x.ycoord()) && ly>y.ycoord()) return false;
    return true;
  }
```

```
  else{
    point p;
    l.intersection(line(), p);
    double lx = p.xcoord();
    if ((lx<x.xcoord()) && lx<y.xcoord()) return false;
    if ((lx>x.xcoord()) && lx>y.xcoord()) return false;
    return true;
  }
}
```

H.6 Main.cc

```
// Id : Main.cc, v2.11994/06/2300 : 01 : 33geoffExp
#include "Trace.h"
#include "Annealer.h"
#include "ESTPProblem.h"

int trace;

#ifdef GRAPHICS
window* Wp;
#endif

main(){

  trace=read_int();

#ifdef GRAPHICS
  window W(600,625);
  W.init(-0.1,1.1,-0.1);
  W.set_show_coordinates(false);
  W.set_node_width(3);
  W.set_mode(src_mode);
  Wp=&W;
#endif

  Annealer A;
  ESTPProblem P;

  A.solve(P);

#ifdef GRAPHICS
  W.acknowledge(" Done!  ");
#endif

}
```

H.7 link.h

```
// Id : link.h, v2.21994/08/2307 : 56 : 08geoffExp
#ifndef MYLINK_H
#define MYLINK_H

// An object used as a list element in the minimum spanning tree list
```

```
// The members are the indicies of the points linked

#include "DT.h"

class link{
public:
  int s;
  int f;
public:
  link(){};
  link(int i, int j){
    s=i;
    f=j;
  }
  link(const link& l){
    s=l.s;
    f=l.f;
  }
  friend void Print(const link& l, ostream& out=cout);
  friend void Read(const link& l, istream& in=cin);
  friend int compare(const link& l, const link& m);
  LEDA_MEMORY(link)
  };
inline void Print(const link& l, ostream& out=cout){
  out ≪ string("[%3d]====[%3d]", l.s, l.f);
}
inline void Read(const link& l, istream& in=cin){};
inline int compare(const link& l, const link& m){
  return 0;
}
LEDA_TYPE_PARAMETER(link)

// A similar object but containing the triangulation items
// of the link points

class DTlink{
public:
  DT_item s;
  DT_item f;
public:
  DTlink(){};
  DTlink(DT_item i, DT_item j){
    s=i;
    f=j;
  }
  DTlink(const DTlink& l){
    s=l.s;
    f=l.f;
  }
  friend void Print(const DTlink& l, ostream& out=cout);
  friend void Read(const DTlink& l, istream& in=cin);
  friend int compare(const DTlink& l, const DTlink& m);
  LEDA_MEMORY(DTlink)
  };
inline void Print(const DTlink& l, ostream& out=cout){};
inline void Read(const DTlink& l, istream& in=cin){};
inline int compare(const DTlink& l, const DTlink& m){
```

```
  return 0;
}
LEDA_TYPE_PARAMETER(DTlink)

#endif
```

H.8 Point.h

```
// Id : Point.h, v2.21994/08/2307 : 55 : 57geoffExp
#ifndef MYPOINT_H
#define MYPOINT_H

// An object used as an array element to keep track of the triangulation
// item of a point

#include <LEDA/point.h>

#include "DT.h"

class Point{
public:
  point pt;
  int dg;
  DT_item dt;
public:
  Point(point p, DT_item d){
    pt=p;
    dg=0;
    dt=d;
  }
  ~Point(){};
  Point(const Point& P){
    pt=P.pt;
    dg=P.dg;
    dt=P.dt;
  }
  friend void Print(const Point& p, ostream& out=cout);
  friend void Read(Point& p, istream& in=cin);
  friend int compare(const Point& p, const Point& q);
  LEDA_MEMORY(Point)
};
inline void Print(const Point& p, ostream& out=cout){}
inline void Read(Point& p, istream& in=cin){}
inline int compare(const Point& p, const Point& q){
  return 0;
}
LEDA_TYPE_PARAMETER(Point)
typedef Point *pPoint;
inline void Print(pPoint& p, ostream& out=cout){};
inline void Read(pPoint& p, istream& in=cin){};
inline int compare(const pPoint& p, const pPoint& q){
  return 0;
}
LEDA_TYPE_PARAMETER(pPoint)
#endif
```

References

[1] E. H. L. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons, 1989. Reprinted February 1990.

[2] J. E. Beasley and F. Goffinet. A Delaunay triangulation based heuristic for the Euclidean Steiner problem. *Networks*. to appear.

[3] J. E. Beasley. A heuristic for Euclidean and rectilinear Steiner problems. *European Journal of Operational Research*, 58:284–292, 1992.

[4] V. Černy. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimisation Theory and Applications*, 45:41–51, 1985.

[5] S. Chang. The generation of minimal Steiner trees with a Steiner topology. *Journal of the Association of Computing Machinery*, 19(4):699–711, 1972.

[6] E. J. Cockayne and D. E. Hewgill. Exact computation of Steiner minimal trees in the plane. *Information Processing Letters*, 22:151–156, 1986.

[7] E. J. Cockayne and D. E. Hewgill. Improved computation of plane Steiner minimal trees. *Algorithmica*, 7:219–229, 1992.

[8] R. Courant and H. Robbins. *What is Mathematics?* Oxford University Press, New York, 1941.

[9] K. Dowsland. Hill-climbing, simulated annealing and the Steiner problem in graphs. *Engineering Optimisation*, 17:91–107, 1991.

[10] D.-Z. Du and F. K. Hwang. The state of art on Steiner ratio problems. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, pages 163–191. World Scientific Publishing Company, 1992.

[11] R. W. Eglese. Simulated annealing: a tool for operational research. *European Journal of Operational Research*, 46:271–281, 1990.

[12] S. Fortune. Voronoi diagrams and Delaunay triangulations. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, pages 163–191. World Scientific Publishing Company, 1992.

[13] M. R. Garey and D. S. Johnson. *Computers and Intracibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[14] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing Steiner minimal trees. *SIAM Journal Applied Mathematics*, 32:835–859, 1977.

[15] E. N. Gilbert. Minimum cost communication networks. *The Bell System Technical Journal*, 46:2209–2227, 1967.

[16] J. Hesser, R. Männer, and O. Stucky. Optimisation of Steiner trees using genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 231–236, San Mateo, California, 1989. Morgan Kaufmann.

[17] F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22:55–89, 1992.

[18] F. K. Hwang, G. D. Song, G. Y. Ting, and D.-Z. Du. A decomposition theorem on Euclidean Steiner minimal trees. *Discrete and Computational Geometry*, 3:367–382, 1988.

[19] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problems*. North-Holland, Amsterdan, 1992.

[20] F. K. Hwang. A linear time algorithm for full Steiner trees. *Operations Research Letters*, 5:235–237, 1986.

[21] F. K. Hwang. Private correspondence. 1993.

[22] A. Kapsalis, V. J. Rayward-Smith, and G.D. Smith. Solving the graphical Steiner tree problem using genetic algorithms. *Journal of the Operational Research Society*, 14(2):397–406, 1993.

[23] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 13 May 1983.

[24] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, 1987.

[25] P. J. M. van Laarhoven. *Theoretical and computational aspects of simulated annealing*. Centrum voor Wiskunde en Informatica, 1988.

[26] S. Lin and B. W. Kernighan. An effective heuristic for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[27] M. Lundy. Aplications of the annealing algorithm to combinatorial problems in statistics. *Biometrika*, 72(1):191–198, 1985.

[28] Z. A. Melzak. On the problem of Steiner. *Canadian Mathematical Bullentin*, 4:143–148, 1961.

[29] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[30] S. Näher. *LEDA User Manual Version 3.0*. Max-Planck-Institut für Informatik, Im Stadtwald D-6600 Saarbrücken, 1992.

[31] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*, pages 8–11. John Wiley & Sons, New York, 1993.

[32] G. Reinelt. TSPLIB - a travelling salesman problem library. *ORSA Journal of Computing*, 3(4):376–384, 1991.

[33] T. K. Sarkar. OPRE451 Lecture Notes (1993): The Euclidean Steiner tree problem. Institute of Statistics and Operations Research, Victoria University of Wellington, New Zealand.

[34] C. Schiemangk. Thermodynamically motivated simulation for solving the Steiner tree problem and the optimisation of interacting path systems. In A. Iwainsky, editor, *Optimisation of Connection Structures in Graphs*, pages 91–120. Central Institute of Cybernetics and Information Processes, Academy of Sciences of the German Democratic Republic, 1985.

[35] J. M. Smith and J. S. Liebman. Steiner trees, Steiner circuits and the interference problem in building design. *Engineering Optimization*, 4:15–36, 1979.

[36] J. M. Smith and P. Winter. Computational geometry and topological network design. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, pages 287–385. World Scientific Publishing Company, 1992.

[37] J. M. Smith, D. T. Lee, and J. S. Liebman. An $O(n \log n)$ heuristic for Steiner minimal tree problems on the Euclidean metric. *Networks*, 11:23–39, 1981.

[38] J. M. Smith. Private correspondence. 1994.

[39] D. Trietsch and F. K. Hwang. An improved algorithm for Steiner trees. *SIAM Journal of Applied Mathematics*, 50:244–263, 1990.

[40] D. Trietsch. Augmenting Euclidean networks—The Steiner case. *SIAM Jounal of Applied Mathematics*, 45(5):855–860, 1985.

[41] D. Trietsch. Interconnecting networks in the plane: The Steiner case. *Networks*, 20:93–108, 1990.

[42] P. Winter. The Steiner problem. Master's thesis, Institute of Datalogy, Copenhagen, Denmark, 1981.

[43] P. Winter. An algorithm for the Steiner problem in the Euclidean plane. *Networks*, 15:323–345, 1985.