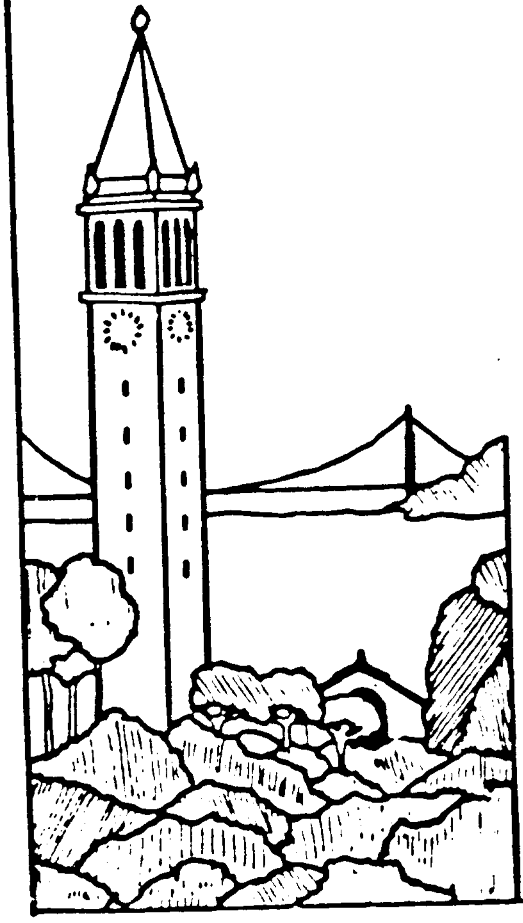
A Greedy Heuristic for the Rectilinear Steiner Tree Problem

Marshall Bern and *Marcio de Carvalho*



Report No. UCB/CSD 87/306

September 1986

Computer Science Division (EECS)
University of California
Berkeley, California 94720

A GREEDY HEURISTIC FOR THE RECTILINEAR STEINER TREE PROBLEM

Marshall W. Bern[*]

Marcio de Carvalho[†]

Computer Science Division

University of California -- Berkeley

ABSTRACT

Clark Thompson recently suggested a very natural "greedy" heuristic for the rectilinear Steiner problem (RSP), analogous to Kruskal's algorithm for the minimum spanning tree problem. We study this heuristic by comparing the solutions it finds with rectilinear minimum spanning trees. We first prove a theoretical result: on instances of RSP consisting of a large number of random points in the unit square, Thompson's heuristic produces a tree of expected length some fraction shorter than a minimum spanning tree. The second part of this paper studies Thompson's heuristic experimentally and finds that it gives solutions about 9% shorter than minimum spanning trees on medium size problems (40-100 nodes). This performance is very similar to that of other RSP heuristics described in the literature.

* Supported in part by NSF grant MCS-8311422

† Supported in part by CAPES grant 3817/82-4

1. Introduction

Assume we have a set V consisting of N points on the plane. A *Steiner tree* for V is a tree that contains these N points as a subset of its vertex set. A *rectilinear Steiner tree* is a Steiner tree for V which contains only horizontal and vertical line segments. The *rectilinear Steiner problem* (RSP) is to find a rectilinear Steiner tree of minimum overall length. This problem bears a close kinship to the problem of finding a rectilinear minimum spanning tree (RMST); the crucial difference is that a solution to the Steiner problem may include extra vertices, called *Steiner points*, if their inclusion helps reduce the overall length of the tree. Efficient algorithms for solving RMST are well-known [Kr, Pr]; RSP, on the other hand, is NP-complete [GJ], hence it is unlikely that any polynomial-time algorithm exists for its exact solution. Researchers have discovered polynomial-time algorithms for some special cases of RSP [AGH], and have devised a number of heuristic algorithms for the general problem [Se, ML, MLL, H78, H79, KoSh]. Hwang [H76] proved that for any set of points in the plane the length of a rectilinear minimum spanning tree is no more than 3/2 the length of a rectilinear Steiner minimal tree.

Our interest in RSP stems from its application to wire layout problems in VLSI circuit design, specifically wiring combinational circuits in gate arrays. For this application, it is desirable to have (suboptimal) solutions to RSP with small overall wire length and a small number of wire bends. Moreover, the randomized rounding method of Raghavan and Thompson [RT] provides motivation for an algorithm that can produce a number of alternative solutions to RSP. This method requires as input a number of alternative wirings, and then uses a randomization technique to produce a routing of small width, that is, with a small number of wires running in parallel along any one gate array channel.

Clark Thompson [Th] proposed the following natural RSP heuristic: start by connecting the two closest nodes with a wire, either straight or "L-shaped" (that is, the union of one horizontal and one vertical line segment), then in each subsequent stage, "greedily" add the shortest wire, straight or L-shaped, node-to-node, node-to-wire, or wire-to-wire, that reduces the number of connected components. Stop when the number of connected components reaches one. As stated, this heuristic allows a free choice of two possible orientations for each L-shaped wire. By varying this choice, either randomly or according to some deterministic rule, a number of alternative solutions may be found.

A literature search revealed that similar, though not identical, heuristics have been tried in the past [Se, MLL, LBH, H78, H79]. The major difference is that the heuristics in the literature use an approach analogous to Prim's RMST algorithm [Pr], rather than to Kruskal's RMST algorithm [Kr]. That is, they grow a tree from a single source node by successively adding in closest nodes, rather than by growing a forest of connected components. The Prim approach typically produces a heuristic with a faster running time, since wire-to-wire distances are never considered.

In this paper, we investigate Thompson's heuristic, first theoretically, and then experimentally. We focus on the question of wire length, though the experiments in section 3 also offer some data on the number of Steiner points and wire bends. Throughout this paper, we compare the Steiner trees produced by Thompson's heuristic to rectilinear minimum spanning trees. In the theoretical section we prove that trees found by a slightly modified version of Thompson's heuristic are asymptotically some fraction shorter than minimum spanning trees. In the experimental section, we show that Thompson's heuristic performs about as well as the published heuristics. Unfortunately, its running time is probably not as fast.

2. Theoretical Analysis

Theoretical analysis of Thompson's algorithm evolved on two fronts: deterministic and probabilistic. The most important deterministic result is that Thompson's algorithm will never give a longer tree than the minimum spanning tree. Our probabilistic result is the following: in a model in which nodes are distributed at random according to a Poisson process with intensity N on the unit square, for N large enough the expected length of a solution given by a variant of Thompson's algorithm is at least .098% shorter than the expected length of a minimum spanning tree. Our analysis is not at all sharp, so that the true fraction is likely to be much higher than .098%.

2.1. Deterministic Analysis

Kruskal's classical "greedy" algorithm for RMST starts by placing a wire between the two closest nodes (rectilinear distance) in V. In each subsequent step, a wire is placed between the two closest nodes that are in different connected components. The original Thompson's algorithm presented in section 1 is analogous to Kruskal's algorithm for RMST [Kr] in its greedy choice of wiring. For the purposes of analysis, we now introduce a modified version of Thompson's algorithm that holds even more closely to Kruskal's algorithm. This version stipulates that at the i-th stage of Thompson's algorithm, for $i = 1, 2, \cdots, N-1$, the vertices in the connected components of the growing forest must be exactly the same vertices as those at the i-th stage of Kruskal's greedy minimum spanning tree algorithm. The two components to connect are chosen by minimum node-to-node distances, but the wire laid down to connect them may attach to previously laid wires on either or both ends.

Kruskal's Algorithm

```
while ∃ more than one connected component do
    find nodes $u$ and $v$ in distinct components with min rectilinear separation
    join $u$ to $v$ with a minimum length rectilinear wire
    od
```

Original Thompson's Algorithm

```
while ∃ more than one connected component do
   find the two connected components $X$ and $Y$ with min rect separation
   join $X$ to $Y$ with a mininimum length rectilinear wire
   od
```

Modified Thompson's Algorithm (Heuristic C)

```
while ∃ more than one connected component do
   find nodes $u$ and $v$ in separate components with min rectilinear separation
   join $u$'s component to $v$'s component with a minimum length rectilinear wire
   od
```

Figure 1 demonstrates that Thompson's original method and the modified method are, indeed, distinct methods: the Steiner trees they create may be different. In this figure original points (nodes) are indicated by boxes, and Steiner points are indicated by circles; the order in which wires are laid down is shown by small numbers next to the wires. In either version, one has a free choice of two possible orientations for L-shaped wires. In Figure 1, all L-shaped wires attach to their uppermost vertex with a horizontal line segment.

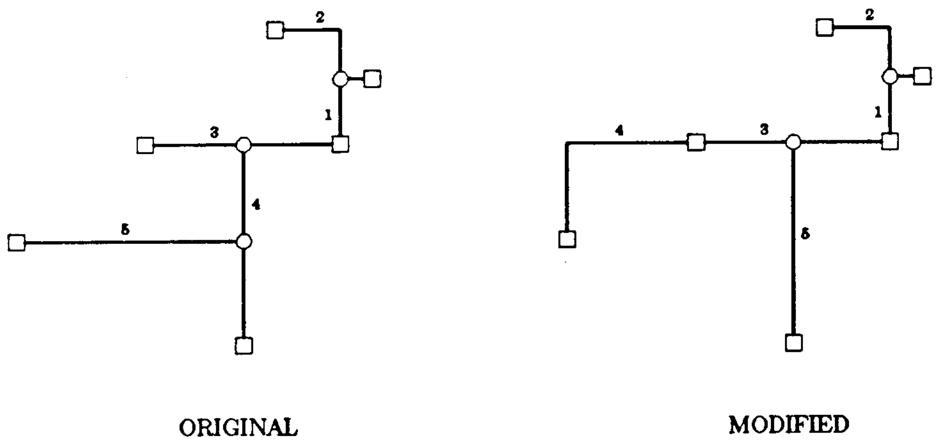


Figure 1. Steiner trees produced by original and modified Thompson's algorithms.

Theorem 1. Both versions of Thompson's algorithm dominate minimum spanning tree.

Proof. The easy one first: it is clear that the modified algorithm dominates minimum spanning tree since at each stage it chooses a wire no longer than the wire chosen by Kruskal's algorithm and connects the same two components.

To prove that the original algorithm dominates minimum spanning tree, we need two technical definitions and an invariant. In the following discussion

Steiner forest refers to a forest on the same N nodes (and any number of Steiner points) on which we are trying to find a Steiner tree. We shall call a set of rectilinear wires C a *cutset* with respect to Steiner forest F if C is the set of all rectilinear wires with exactly one endpoint in a single connected component of F. Wires in a cutset may be node-to-node or may have one or both endpoints on wires in F; that is, we consider the connected components of F as including nodes and wires. (Notice that if we are not constraining Steiner points to grid points, a cutset may be infinite.) We shall call a Steiner forest F *promising* if it is possible to add node-to-node wires to F to form a Steiner tree of length no greater than the length of a minimum spanning tree. The forest with no edges is obviously promising. We assert that if F is a promising forest and e is the shortest wire in a cutset with respect to F, then F with e added is promising. Consider a Steiner tree T no longer than a minimum spanning tree consisting of F with only node-to-node wires added. To this Steiner tree we could add wire e and form a cycle. (If T already contained e, we would be done.) This cycle would necessarily contain a wire f, $f \neq e$, where f is one of the node-to-node wires in the cutset of which e is a shortest wire. Removing f would result in a new Steiner tree of length no longer than the length of T. We have now proved the assertion.

A single stage in the original Thompson's algorithm adds the shortest wire it can add without forming a cycle. Calling one of the components joined by this wire X, the set of all rectilinear wires joining X to another component is a cutset with respect to the Steiner forest already grown by the algorithm. We can now apply the assertion above to conclude that after each stage of the original algorithm, including the last stage, the Steiner forest is promising. Thus the Steiner tree produced by this algorithm is no longer than a minimum spanning tree. $\Box$

Theorem 2. Neither version of Thompson's algorithm dominates the other.

Proof. Figure 1 gives an instance of the rectilinear Steiner tree problem in which the original algorithm yields a shorter solution than the modified algorithm. Figure 2 gives an instance in which the modified algorithm outperforms the original. We again used the rule of attaching to uppermost nodes horizontally, but examples like Figures 1 and 2 can be concocted for any tie-breaking rule. $\Box$

2.2. Probabilistic Analysis

For the purpose of analysis, we choose to use the modified version of Thompson's algorithm in this section. We are interested in random instances of the Steiner tree problem. The most intuitively appealing probability space contains instances in which N nodes are distributed independently and identically, uniformly on $[0, 1] \times [0, 1]$. (Since our focus is on asymptotics, we do not limit our attention to cases in which the nodes are at grid points.) We find it more

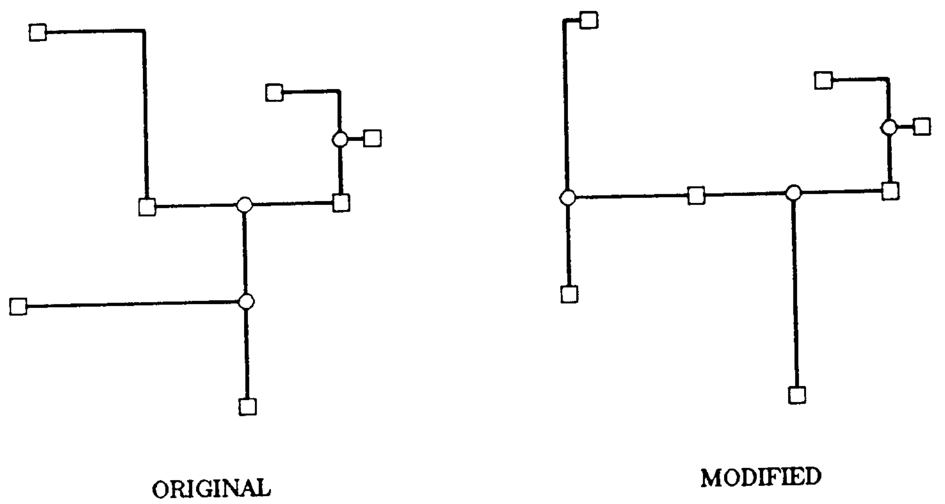


Figure 2. An instance in which the modified Thompson's algorithm outperforms the original.

convenient, however, to work in a Poisson model. Typically, theorems proven in the Poisson model can be transported back to the "exactly-N" model [St, KS], but we could not find a way to do this with our results. Nevertheless, our results have implications for the exactly-N model.

The Poisson probability space is convenient because it avoids conditioning due to the constraint that there are exactly N nodes. We first consider a Poisson process with intensity 1 in a square of side $\sqrt{N}$. In the Poisson model:

(1) The probability of k nodes appearing in any region of area A is $e^{-A}A^k/k!$.

(2) The conditional distribution of nodes in any region of area A given that exactly k nodes fall in A is joint uniform (that is, the k nodes are independent and identically distributed uniformly on that region).

We further assume that N is large enough that we can inscribe two smaller concentric squares within the large $\sqrt{N} \times \sqrt{N}$ square, such that the borders between the outermost square S_1 and the middle square S_2 and between S_2 and the innermost square S_3 are each of width 3. This division is necessary to handle certain boundary effects. Of course there is nothing special about the number 3; any larger number would serve equally well. Finally a word on notation: $P[\cdot]$ denotes the probability of an event, $E[\cdot]$ denotes the expectation of a random variable, and $d(u,v)$ denotes the rectilinear (or Manhattan) distance between nodes u and v. For any node v, define $n(v)$ to be v's nearest neighbor; that is, if u is another node, $u \neq v$, $d(n(v),v) \leq d(u,v)$.

Our strategy in showing that the modified Thompson's algorithm (hereafter called Heuristic C for "Clark") asymptotically outperforms the minimum spanning tree heuristic is to identify a situation in which Heuristic C makes a better choice of wire than Kruskal's algorithm, and then show that this situation recurs frequently. The situation which we shall use is shown in Figure 3. The essential ingredients are that one of the two connected components being joined contains exactly 2 vertices, and that the new wire extends from the size 2 component (hereafter called a *2-component*) in the same direction that the wire within the 2-component makes an L-bend. Kruskal's algorithm will at some stage form a 2-component containing vertices u and v if and only if $n(v) = u$ and $n(u) = v$. The next lemma calculates the chance of a given node participating in a 2-component.

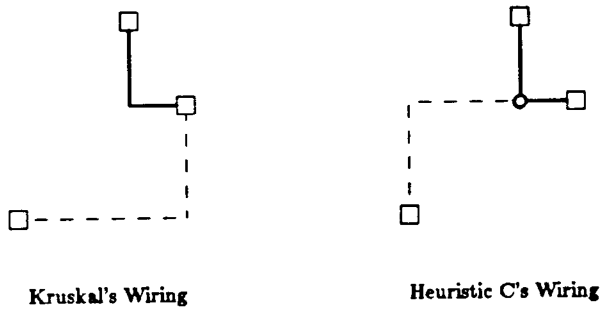


Figure 3. Wiring a 2-component.

Lemma 1. For each node v in S_3, $P[n(v)$ is in S_2 and $n(n(v)) = v] \geq .57$.

Proof. The probability that $d(v,n(v)) \geq x$ is equal to the probability that there are no nodes (other than v) in a rectilinear ball of radius x around v, which is equal to e^{-2x^2} by property (1) of the Poisson process. Differentiating, we obtain the probability density function for $d(v,n(v))$ of $4xe^{-2x^2}$.

Now let u be $n(v)$. For any v in S_3, the probability that $d(u,v) \leq 3$ is larger than .999, since $P[d(u,v) \geq 3] = e^{-18} < .001$, so assume we are in the case that $d(u,v) \leq 3$. Notice that in this case u is necessarily within S_2. The node u must be located on the boundary of a rectilinear ball of radius $d(u,v)$ about v, as in Figure 4. All of this ball is within the larger square S_1, and no nodes other than v are within the interior of the ball. If $n(u) \neq v$, some node w must be within a ball of radius $d(u,v)$ about u. It is not hard to calculate that the area of the shaded region R in Figure 4 varies linearly from $(d(u,v))^2$ to $(3/2)(d(u,v))^2$ depending upon the location of u on a side of the boundary of the rectilinear ball. We overestimate the area of the shaded region, and hence underestimate the chance of $n(u) = v$, by assuming the area to always be $(3/2)(d(u,v))^2$. Continuing the calculation, $P[\nexists$ a node w in $R] \geq \int_0^3 4xe^{-2x^2} \cdot e^{-3x^2/2}dx$, that is, the integral over x of the instantaneous probability that $d(u,v) = x$ times the conditional probability that there are 0 nodes in R, given that R has area $(3/2)x^2$. Thus, $P[\nexists$ a node w in $R] \geq (4/7)\int_0^3 7xe^{-7x^2/2}dx \geq (4/7)(1 - e^{-63/2}) \geq .571$. So altogether, for each node v in S_3, $P[n(v)$ is in S_2 and $n(n(v)) = v] \geq .999 \times .571$. $\square$

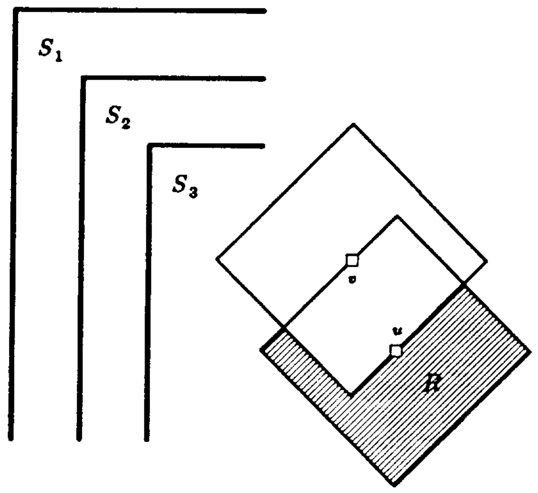


Figure 4. If there are no nodes in the shaded region R, then $n(u) = v$.

For distinct nodes u and v the probabilities $P[n(n(u)) = u]$ and $P[n(n(v)) = v]$ are not independent. This lack of independence, however, will not turn out to be a problem.

In an execution of Kruskal's minimum spanning tree algorithm, each 2-component must later be wired to some other component. Each of these later wirings is potentially a situation as in Figure 3, that is, a situation in which Heuristic C will save wire. In wirings between two 2-components Heuristic C has two (nonindependent) chances to save, thus we may associate the potential savings with the 2-components themselves, rather than with the wire joining the 2-components.

Let us look at a 2-component more closely in Figure 5. Nodes u and v are connected by an L-shaped wire; c_{uv} denotes the corner point of this wire. (We ignore the case in which u and v can be connected with a single horizontal or vertical line. This case occurs with probability 0 when nodes are not constrained to be on grid points.) Further assume that both u and v are in S_2; such a 2-component we shall call an *interior* 2-component. We extend our notation so that $n(u,v)$ denotes the node nearest the set of nodes $\{u,v\}$, that is, node w is called $n(u,v)$ if for any other node y, $y \neq u$, $y \neq v$, we have $\min\{d(w,u),d(w,v)\} \leq \min\{d(y,u),d(y,v)\}$. If u and v form an interior 2-component during the execution of Kruskal's algorithm, then $w = n(u,v) = n(u,n(u))$ is the next node to be wired to the component containing u and v. (Of course, w need not be isolated as in Figure 5. In fact, Heuristic C may not even send a wire to w if the connected component containing w contains a closer corner point. The fact is that Heuristic C will never use a wire longer than the distance from w to the c_{uv}.) Node w must be outside the union of balls of radius $d(u,v)$ around u and v since $n(u) = v$ and $n(v) = u$. We would like to estimate the probability that $n(u,v) = w$ is in the region where Heuristic C saves the most wire. This region, which we shall call R_{uv}, is the intersection of the square S_1 with the quadrant of the plane centered at c_{uv} that includes neither u nor v along its boundary.

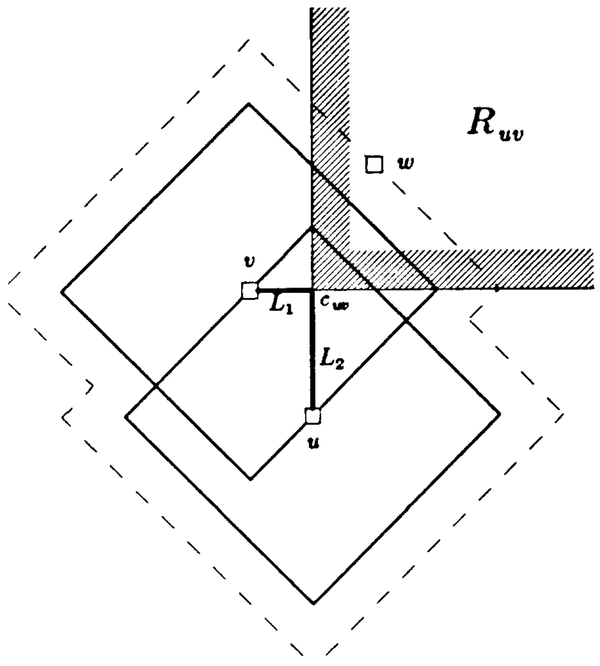


Figure 5. Heuristic C saves at least $\min\{L_1,L_2\}$ if $n(u,v)$ is in R_{uv}.

The argument goes as follows: w must be at some distance $d = \min\{d(w,u),d(w,v)\}$ from the set $\{u,v\}$. The probability that d is less than 3 is greater than .999, so assume this is the case. (The fact that $P[d \le 3] > .999$ can be seen by noticing that the probability that there exist no nodes within distance x of a 2-component is less than the probability that there exist no nodes within distance x of a single node.) Since the knowledge that w is $n(u,v)$ in no way conditions the direction of w, all points at distance d are equally likely. Thus, we need only determine what fraction of a line at distance d from $\{u,v\}$ lies within R_{uv}. An easy calculation shows that the length of the boundary of the union of the balls of radius d around u and v that falls within R_{uv} is $\sqrt{2}(d - \min\{L_1,L_2\})$, where L_1 is the horizontal separation and L_2 the vertical separation of u and v. The total perimeter of the balls is between $5\sqrt{2}d$ (achieved when $L_1 = L_2$ and approached for any L_1 and L_2 when d is relatively large) and $6\sqrt{2}d$ (approached when L_1 or L_2 is close to 0). Thus, the fraction of the line within R_{uv} is at least $\sqrt{2}(d - \min\{L_1,L_2\}) / 6\sqrt{2}d$, Recalling that $d \ge L_1 + L_2$, we see that this last quantity is at least 1/12. We have thus proven the following:

Lemma 2. Assume nodes u and v form an interior 2-component. Then for horizontal and vertical separations $L_1 \geq 0$ and $L_2 \geq 0$, $P[n(u,v)$ is in $R_{uv}] \geq 1/12 - .001$. □

One more estimation lemma is necessary:

Lemma 3. If L_1 and L_2 are the lengths of the horizontal and vertical wire segments in an interior 2-component, then $E[\min\{L_1,L_2\}] \geq .06$.

Proof. As in the proof of Lemma 1, we consider v's location fixed and u's location to be distributed on a ball of radius $d(u,v)$ about v. The random variable $\min\{L_1,L_2\}$ varies between 0 and $(1/2)d(u,v)$ with u's position on a side of the ball. Conditioning on the fact that $n(u) = v$ only increases the likelihood that u is near the middle of a side, and thus that $\min\{L_1, L_2\}$ is nearer $(1/2)d(u,v)$. Thus, $E[\min\{L_1, L_2\} \quad] \geq (1/4)E[d(u,v) \mid u = n(v), v = n(u)] \geq (1/4)\int_0^3 x4x\, e^{-2x^2} \cdot e^{-3x^2/2}\, dx$. We can do this integral by parts to obtain $E[\min\{L_1, L_2\}] \geq (-1/7)\cdot 3\cdot e^{-63/2} + (1/7)\int_0^3 e^{-7x^2/2}dx$. The value of this latter integral can be found by a change of variables ($y = \sqrt{7/2}x$) and a look in any table of the error function. Altogether we see that $E[\min\{L_1, L_2\}] \geq .06$. □

Combining Lemmas 2 and 3, we can conclude that Heuristic C produces an expected wire savings over Kruskal's minimum spanning tree algorithm of about .005 per interior 2-component, or about .0025 per node in an interior 2-component. These savings are not necessarily independent for distinct nodes u and v, but this will not matter for our purposes. We are now in a position to state our main results. Let $\Pi(N)$ denote a set of nodes specified by the unit intensity Poisson process on the square $[0,\sqrt{N}] \times [0,\sqrt{N}]$. Let $C(\Pi(N))$ and $M(\Pi(N))$ denote, respectively, the length of the Steiner tree found by Heuristic C and the length of a minimum spanning tree on $\Pi(N)$.

Theorem 3. $\liminf_{N\to\infty} E[M(\Pi(N)) - C(\Pi(N))] / N \geq .0014$.

Proof. $E[M(\Pi(N)) - C(\Pi(N))]$ is greater than the sum over all nodes v of the probability that v participates in an interior 2-component times the expected wire savings credited to nodes in interior 2-components. Notice that here we are using the fact that expectations add regardless of independence. Now for any node v, $P[v$ is in $S_3] = (1 - (12/\sqrt{N}))^2$, $P[n(v)$ is in S_2 and $n(n(v)) = v \mid v$ is in $S_3] \geq .57$, and the expected wire savings given that v is in S_3, $n(v)$ is in S_2, and $n(n(v)) = v$, is at least .0025. Thus, $E[M(\Pi(N)) - C(\Pi(N))] \geq N\cdot(1 - (12/\sqrt{N}))^2 \cdot .57 \cdot .0025$

$\geq$ .0014N, for sufficiently large N. ▯

We now consider nodes on the unit square. Let $X_1, X_2, \cdots$ be an infinite collection of i.i.d. random variables (that is, nodes) distributed uniformly on $[0,1] \times [0,1]$, and let $\Pi'(N)$ be nodes given by a Poisson process of intensity N on $[0,1] \times [0,1]$. Let $S(X_1, \ldots, X_N)$, $C(X_1, \ldots, X_N)$, and $M(X_1, \ldots, X_N)$ denote, respectively, the length of an optimal Steiner tree, the Steiner tree found by Heuristic C, and the length of a minimum spanning tree on the nodes $X_1, X_2, \ldots, X_N$. Let $S(\Pi'(N))$, $C(\Pi'(N))$, $M(\Pi'(N))$ denote the same functionals on the nodes $\Pi'(N)$.

Steele's work [St] shows that both $S(X_1, \ldots, X_N) / \sqrt{N}$ and $S(\Pi'(N)) / \sqrt{N}$ converge almost surely to β_S, with β_S an unknown constant greater than 0. Similarly, both $M(X_1, \ldots, X_N) / \sqrt{N}$ and $M(\Pi'(N)) / \sqrt{N}$ converge almost surely to β_M. Haimovich and Rinnooy Kan [HRK] show that the asymptotic worst case length of a minimum spanning tree on N nodes in the unit square is $\sqrt{2}\sqrt{N}$, that is, for any sequence of nodes $X_1, X_2, \cdots$, $\limsup_{N \to \infty} M(X_1, \ldots, X_N) / \sqrt{N} \leq \sqrt{2}$. This result obviously gives an upper bound of $\sqrt{2}$ on β_M. There is an upper bound of 1 on β_S [CH, CG]. We know of no published lower bounds on these constants, though it is not hard to show that β_M and β_S are at least $\sqrt{2\pi}/8$ (about .31) by calculating the expected distance between a node and its nearest neighbor. Our experiments in section 3 suggest that β_M is between .7 and .8.

Corollary 1. $\beta_M - .0014 \geq \beta_S$.

Proof. Heuristic C gives a feasible solution to RSP, thus $C(\Pi'(N)) \geq S(\Pi'(N))$. Now Theorem 3 (suitably rescaled for the unit square) implies Corollary 1. ▯

The following corollary gives Heuristic C's wire savings in the form of an expected fraction of the total wire.

Corollary 2. $\liminf_{N \to \infty} E[M(\Pi'(N)) - C(\Pi'(N))] / E[M(\Pi'(N))] \geq .00098$.

Proof. For any $\varepsilon > 0$, we have that $E[M(\Pi'(N)) - C(\Pi'(N))] / E[M(\Pi'(N))] \geq E[M(\Pi'(N)) - C(\Pi'(N))] / \sqrt{N}(\beta_M + \varepsilon)$ for sufficiently large N. Using Theorem 3 (rescaled) and the result of [HRK] that $\beta_M \leq \sqrt{2}$, we obtain Corollary 2. ▯

We now comment on the exactly-N analog of Theorem 3, that is, $\liminf_{N \to \infty} E[M(X_1, \ldots, X_N) - C(X_1, \ldots, X_N)] / \sqrt{N} \geq .0014$. This analog would follow if we could prove a seemingly trivial smoothness condition on the sequence $G(1), G(2), \cdots$, where $G(N) = E[M(X_1, \ldots, X_N) - C(X_1, \ldots, X_N)] / \sqrt{N}$. If

$|G(N+1) - G(N)|$ could be shown to be $o(1/\sqrt{N})$, then the result would follow, using a certain "Tauberian" theorem in the literature [P]. Unfortunately, the behavior of Heuristic C is not simple, and it appears difficult to prove even this "trivial" condition. In any case, we can conclude that $\limsup_{N\to\infty} E[M(X_1, \ldots ,X_N) - C(X_1, \ldots ,X_N)] / \sqrt{N} \geq .0014$.

2.3. Theoretical Conclusions

Our probabilistic analysis leads us to suggest two other heuristics for RSP: the first is a simpler heuristic with the same asymptotic performance guarantee; the second is a less quick heuristic with a potentially better guarantee. The first algorithm would be the same as the modified Thompson's algorithm with the exception that only the corner points of L-shaped wires may be used as Steiner points. In other words, at each stage of the algorithm the connected components are chosen as in Kruskal's algorithm (by shortest node-to-node distance) but the connection is wired to either the closest node or the closest corner point. Ng [N] has used a similar algorithm (the difference being that the two components to connect are chosen by point-to-point distances, where a "point" may be a corner point)) for wiring nets in gate arrays. Ng's algorithm is especially simple to implement; all that is required is to modify an implementation of Kruskal's algorithm to allow the insertion of new points as the algorithm runs.

The second, less quick, heuristic suggested by our work is one in which an initial stage lays down wires along a minimum length perfect matching. A second stage greedily wires the resulting 2-components, as in Thompson's algorithm. In this heuristic, all nodes participate in 2-components rather than just 57% of all nodes. On the other hand, a small price is paid in that the matching may require some relatively long wires that a minimum spanning tree does not.

3. Empirical Analysis

In this section we investigate the performance of Thompson's algorithm on randomly generated problem instances. For simplicity, we let the grid size and the number of nodes in the instance be completely dependent: we generated instances consisting of N nodes in an $N \times N$ grid, with N ranging from 10 to 100. As in the theoretical analysis, we compare the length of a Steiner tree produced by the two versions of Thompson's algorithm with the length of a minimum spanning tree on the same set of nodes. All our programs were implemented in Pascal.

3.1. Implementation

Our presentation will be based on the following high-level algorithm:

```
program Steiner;
   GetNodes;                    { Gets a random problem instance }
   NumberofComponents ← $N$;
   while NumberofComponents > 1 do
     $(x_1,y_1)$, $(x_2,y_2)$ ← Closest;      { Returns and deletes closest pair of elements }
     if $(x_1,y_1)$ and $(x_2,y_2)$ are not in the same component then
        Connect $((x_1,y_1), (x_2,y_2))$;
        NumberofComponents ← NumberofComponents - 1;
     fi
   od
```

Algorithm 1. High-level Steiner Tree Algorithm

Both versions of Thompson's algorithm are embedded in Algorithm 1; their differences are a consequence of different alternatives adopted for the subroutines. In the original version, the routine "Closest" considers both nodes and wires. In the modified version, the routine "Closest" considers only nodes.

We chose to avoid the complexity of handling wires (that is, line segments) using only endpoints. We maintained an $N \times N$ array of characters, representing the grid. (This decision especially simplified printing out graphical representations of the Steiner trees.) Cells in the grid could be in one of five states: empty, node, Steiner point, wire, or corner. Given our simple choice of data structure, a naive implementation of "Closest" and "Connect" would be very slow. For example, if "Closest" were to compute the minimum rectilinear distance over all possible pairs of elements each time it were called, the complexity of the algorithm would be no better than $\Omega(N^5)$, since the number of elements (node cells and wire cells) would be $\Omega(N^2)$ in the worst case, and the number of calls to "Closest" would be at least $N-1$. The number of calls to "Closest" would be much more than $N-1$ if most of the calls returned a pair of elements from the same component.

An efficient implementation of the operations "Closest" and "Connect" is achieved with suitable data structures. Pairs of cells and their distances (the keys) are stored in a d-heap tree which allows us to obtain and delete its minimum value in $O(d\log_d n)$ and to insert a new element in $O(log_d n)$, where d is the maximum number of children a node can have and n is the number of elements in the heap [T]. The parameter d is chosen according to the relative frequency of delete and insert operations; we used $d = 4$, which works fairly well over a wide range of operation mixes. Our heap initially contains only node-to-node distances. As the algorithm proceeds, however, "Closest" updates the heap, adding wire-to-node distances and wire-to-corner distances. Wire-to-corner distances are distances from cells containing a new wire point to cells containing a corner point of a previously placed wire. These distances are the only wire-to-wire node distances that must be remembered, since any wire-to-wire line may be "slid", without changing its length, until it becomes a node-to-wire or corner-to-wire line.

In order to determine whether two elements are in the same component, we make use of a "union-find" data structure [AHU, T]. This structure is used to maintain a collection of disjoint sets allowing the following operations: *union* which merges two sets and *find* which returns the set containing a given element. This structure allows us to perform these operations in almost linear running time, actually $O(m\alpha(m,n))$, where m is the number of operations, n the number of elements and $\alpha(m,n)$ is the inverse of Ackerman's function. The number α is very slow growing, in fact, $\alpha(m,n) \leq 3$ for $n < 2^{16}$; thus for all practical purposes, $\alpha(m,n)$ is a constant not larger than 4 [T].

A final point concerns the decision one has to make when wiring two non-collinear elements; there are two possible choices of where to place the corner point of the L-shaped wire. In our implementation of "Connect", the corner point always has as its x-coordinate the x-coordinate of the first parameter, and its y-coordinate is always the y-coordinate of the second parameter.

Our implementation of Thompson's original algorithm is given in Algorithm 2. The routine "GetNodes" generates a random problem instance. The set "Points" remembers node points and corner points. The routines "Union", "Find", "Deletemin", and "Insert" are as in [T], with a few exceptions. "Deletemin" returns a pair of points as the name of the minimum item; this pair gives the coordinates of the endpoints of the shortest wire. Similarly "Insert" is called with a pair of points as two of its arguments.

We now consider the implementation of the modified Thompson's algorithm. In this version the next component to be connected is determined by closest nodes. Thus, the heap will contain only distances between nodes, not between arbitrary elements (wire and node cells) as in the original version. It will be necessary, however, to define another structure suitable for storing the coordinates of the closest elements of each pair of connected components. A (necessarily symmetric) matrix is used for this purpose. This matrix is indexed by representatives of components; entry i,j contains the coordinates of the closest pair of cells, one of which is from the component with representative node i and the other from the component with representative node j. (Until this point we have referred to nodes only by coordinates; in the actual Pascal implementation, nodes also have unique numbers.) After each "Union" operation, the row (respectively, column) corresponding to the root node is set to the component by component minimum of the rows (respectively, columns) of the old root nodes. The modified routine "Connect" is presented in Algorithm 3; all other modules of Algorithm 2 remain unchanged.

```
program Steiner;
   GetNodes;
   NumberofComponents ← $N$;
   while NumberofComponents > 1 do
      $(x_1,y_1)$, $(x_2,y_2)$ ← Closest;
      if Find($(x_1,y_1)$) $\neq$ Find($(x_2,y_2)$) then
         Connect($(x_1,y_1)$,$(x_2,y_2)$);
         Union($(x_1,y_1)$,$(x_2,y_2)$);
         NumberofComponents ← NumberofComponents - 1;
      fi
   od

procedure GetNodes;
   for $i$ ← 1 to $N$ do
      $x_1$ ← random number between 1 and Gridsize;
      $y_1$ ← random number between 1 and Gridsize;
      for $(x_2,y_2)$ $\epsilon$ Points do
         Insert (heap,$(x_1,y_1)$,$(x_2,y_2)$,dist($(x_1,y_1)$,$(x_2,y_2)$));
        Points ← Points + $(x_1,y_1)$;
      od
   od

function Closest;
   Closest ← Deletemin(heap);

procedure Connect ($(x_1,y_1)$, $(x_2,y_2)$);
   if $x_1 = x_2$ then
      for $y$ ← min $\{y_1,y_2\}$ to max $\{y_1,y_2\}$ do
         for $(x',y')$ $\epsilon$ Points do
            Insert (heap,$(x_1,y)$,$(x',y')$, dist($(x_1,y)$,$(x',y')$));
         od
      od
   else if $y_1 = y_2$ then
      for $x$ ← min $\{x_1,x_2\}$ to max $\{x_1,x_2\}$ do
         for $(x',y')$ $\epsilon$ Points do
            Insert (heap,$(x,y_1)$,$(x',y')$, dist($(x,y_1)$,$(x',y')$));
         od
      od
   else
      Points ← Points + $(x_1,y_1)$
      Connect($(x_1,y_2)$, $(x_2,y_2)$);
      Connect($(x_1,y_2)$, $(x_1,y_1)$);
      Union($(x_1,y_2)$, $(x_1,y_1)$);
```

Algorithm 2. Our implementation of the original Thompson's Steiner Tree Algorithm.

3.2. Running Times

What are the running times of our algorithms? In order to separate the effects of grid size and number of nodes, in this section we assume that our problem instances consist of N nodes in an $M \times M$ grid. Starting our analysis at the beginning, we recall that the original and modified algorithms share a common

```
procedure Connect $((x'_1,y'_1),(x'_2,y'_2))$;
   $(x_1,y_1)$, $(x_2,y_2)$ ← Array [Find($(x'_1,y'_1)$), Find($(x'_2,y'_2)$)];
   Conn $((x_1,y_1),(x_2,y_2))$;

procedure Conn $((x_1,y_1),(x_2,y_2))$;
   if $x_1 = x_2$ then
      for $y$ ← min $\{y_1,y_2\}$ to max $\{y_1,y_2\}$ do
         for $(x',y')$ ε Points do
            if dist($(x_1,y),(x',y')$) <
                     dist (Array [Find($(x_1,y)$), Find($(x',y')$)]) then
               Array [Find($(x_1,y)$), Find($(x',y')$)] ← $(x_1,y)$, $(x',y')$;
            fi
         od
      od
   else if $y_1 = y_2$ then
      for $x$ ← min $\{x_1,x_2\}$ to max $\{x_1,x_2\}$ do
         for $(x',y')$ ε Points do
            if dist($(x,y_1),(x',y')$) <
                     dist (Array [Find($(x,y_1)$), Find($(x',y')$)]) then
               Array [Find($(x,y_1)$), Find($(x',y')$)] ← $(x,y_1)$, $(x',y')$;
            fi
         od
      od
   else
      Points ← Points + $(x_1,y_1)$;
      Conn($(x_1,y_2),(x_2,y_2)$);
      Conn($(x_1,y_2),(x_1,y_1)$);
      Union($(x_1,y_2),(x_1,y_1)$);
   fi
```

Algorithm 3. Pseudocode for the modified algorithm's "Connect"

input phase. As each point's coordinates are generated, the distances from that point to all previous points are computed, and the distances are stored in a heap. At the end, $O(N^2)$ distances will be stored in the heap. Initial insertion time is $O(\log N)$, so the total time for all initial insertions is $O(N^2 \log N)$.

Now consider the rest of the original algorithm. For each cell on a wire placed by "Connect", the algorithm computes the distances to all nodes and previously placed corner points and inserts these distances into the heap. The number of nodes and corner points is $O(N)$, and an L-shaped wire in an $M \times M$ grid has length $O(M)$, so the number of insertions is $O(MN)$ for each call to "Connect". Counting all $N-1$ calls, the maximum size of the heap is seen to be $O(MN^2)$. Thus, each insertion takes time $O(\log(MN^2))$ which is $O(\log M)$, so we can conclude that all calls to "Connect" take total time $O(MN^2 \log M)$. Using the fact just noted that the maximum size of the heap is $O(MN^2)$, we see that all the calls to "Closest" take total time $O(MN^2 \log M)$. Finally, the union-find part of the algorithm is relatively fast: only MN cells can be "unioned" together (the size of the entire tree), thus, the time for all calls to "Union" and "Find" is practically $O(MN)$. Overall, we can conclude that our implementation of Thompson's original algorithm has time complexity $O(MN^2\log M)$.

The modified algorithm turns out to be slightly speedier. In this case, too, the innermost loops of "Connect" are the dominating influence. Because the cardinality of the set "Points" is $O(N)$, the number of cells on a wire is $O(M)$, and the number of wires is $O(N)$, the lines within these loops are executed $O(MN^2)$ times. Each pass through one of these loops executes a comparison, an assignment, and two "Finds" on a union-find data structure on which no more than MN "Union" calls have been done. Thus, the modified algorithm has time complexity $O(MN^2)$.

Both these running times are "pseudo-polynomial" in the size of the problem instance; that is, they are polynomial in the length of the problem instance only if node coordinates are given in unary rather than binary notation. Note, however, that fully polynomial implementations of these algorithms are not hard to write.

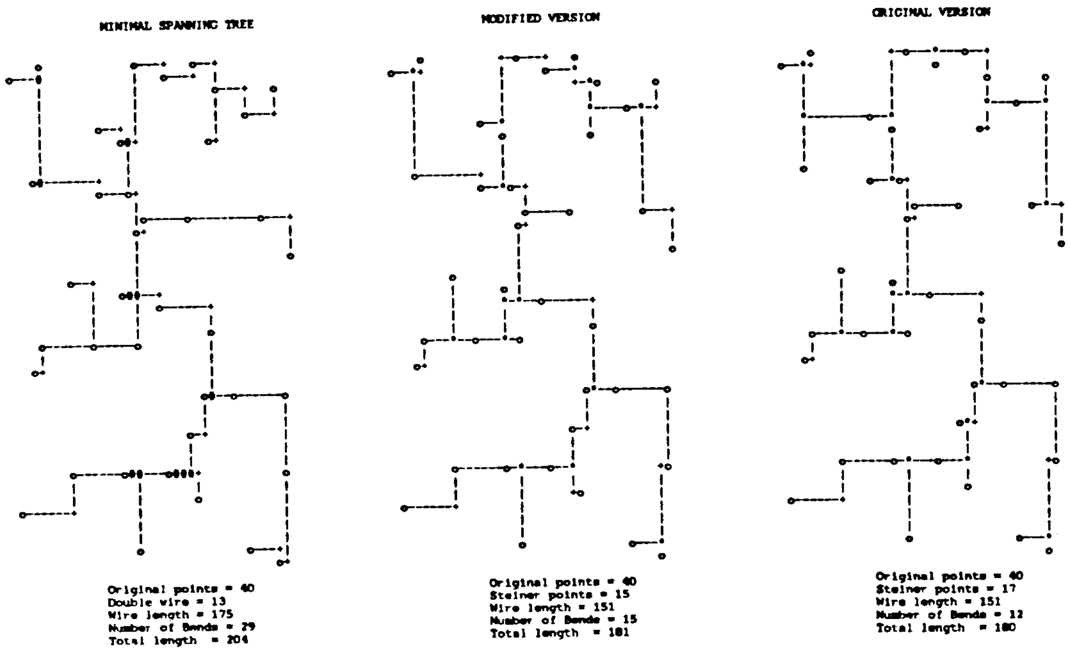


Figure 6. Trees produced by the algorithms described in this paper.

3.3. Fixed Size Problems

The size chosen for these experiments was 40 nodes in a 40×40 grid, which we believed to be realistic for VLSI routing problems. The algorithm running on a VAX 11/785 took only 0.33 seconds of processor time to solve a problem of this size, making it possible to run a large number of problem instances. For each instance, 40 coordinates are randomly generated and three trees containing these nodes are obtained: the minimum spanning tree and the two Steiner trees, one for each version of Thompson's algorithm. Figure 6 shows an example; other examples are included in Appendix A. (In these figures, circles represent nodes, + denotes a corner point, * denotes a Steiner point, and # denotes a cell that contains more than one wire in the case of the minimum spanning tree.) For each trial, we recorded the fraction of wire saved by each of the two versions of Thompson's algorithm relative to the length of a minimum spanning tree.

Algorithm	Mean (%)	σ
Original	9.298	2.17
Modified	9.365	2.12

Figure 7. Comparative Data from 5,000 cases

Figure 7 shows the means and standard deviations of the values (expressed as per cents) obtained. Both algorithms on the average produced Steiner trees more than 9% shorter than minimum spanning trees. In Figures 8 and 9 we show histograms of the algorithms' peformance. Note the high concentration around the mean, which suggests that the algorithm is robust.

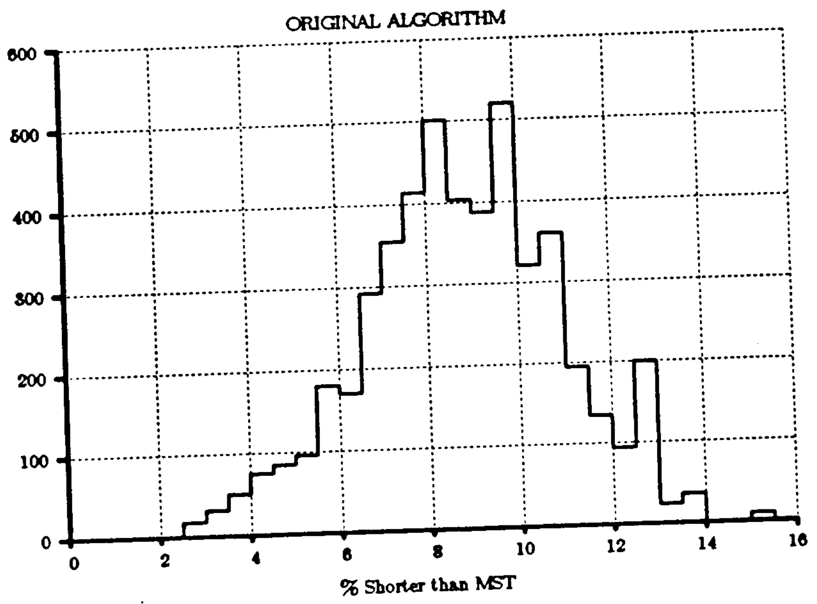


Figure 8. Histogram of the fractional wire savings of the original algorithm.

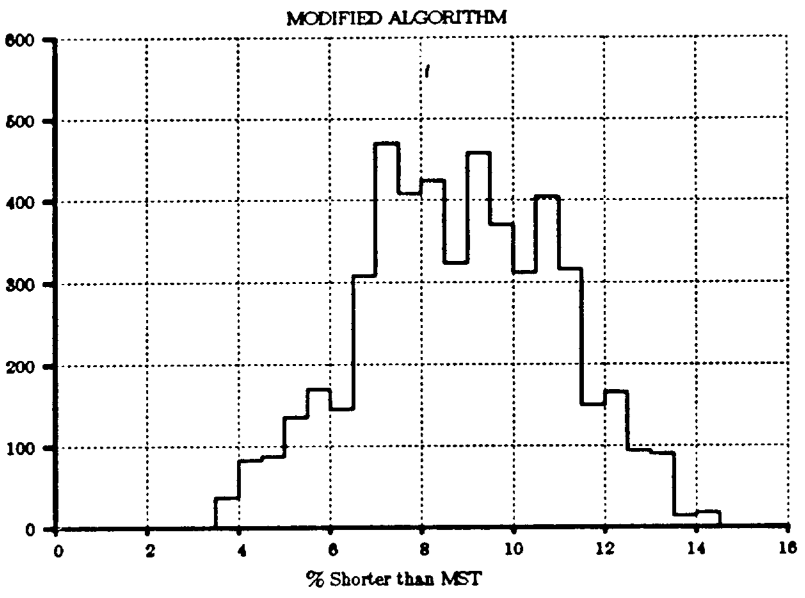


Figure 9. Histogram of the fractional wire savings of the modified algorithm.

3.4. Variable Size Simulation

The main goal of this simulation was to acquire some feeling about the performance of the algorithms as the size of the problem increases. The running times in this experiment were much larger than in the previous case, so only 500 instances were analyzed, all of them using the modified algorithm. The problem instances are all of the form N random points in a grid of size N by N. Figure 10 presents a plot of the data (labeled "Modified") obtained from this simulation, showing N along the x-axis, and the wire savings of the Steiner tree relative to the minimum spanning tree along the y-axis. The thin lines on either side of the data line indicate the boundaries of a 95% confidence interval. We also plot the percent of wire that is "doubled" in the minimum spanning tree; this wire would be saved even by the simplistic algorithm of computing a minimum spanning tree and then removing one wire of each doubled wire. We believe that the true asymptotic advantage of Thompson's algorithm is lower than 9%, though, of course, not so low as is indicated by our theoretical results.

Finally, Figure 11 shows the performance of Thompson's algorithm according to other criteria-- the numbers of wire bends and Steiner points. Again problems of size N were N randomly placed nodes in an $N \times N$ grid. In this experiment both the number of bends and the number of Steiner points grow just slightly faster than linearly with N. This is because nodes are sparser for larger N. (The chance is $1/N$ that a cell is occupied by a node.)

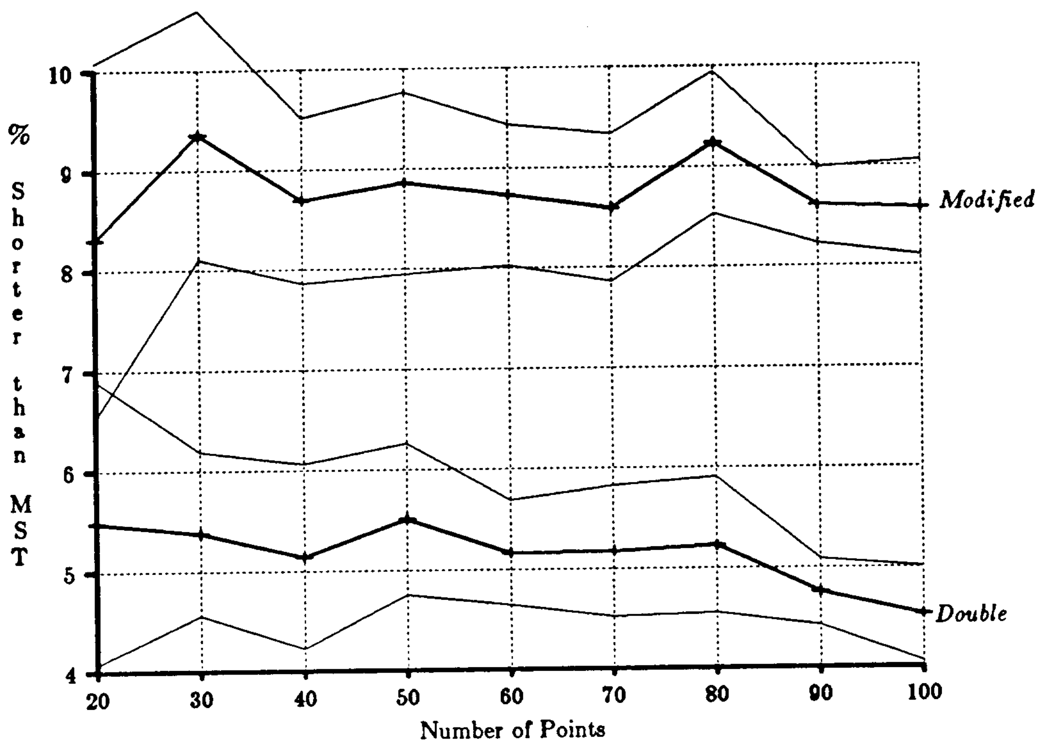


Figure 10. Wire Savings as a function of problem size.

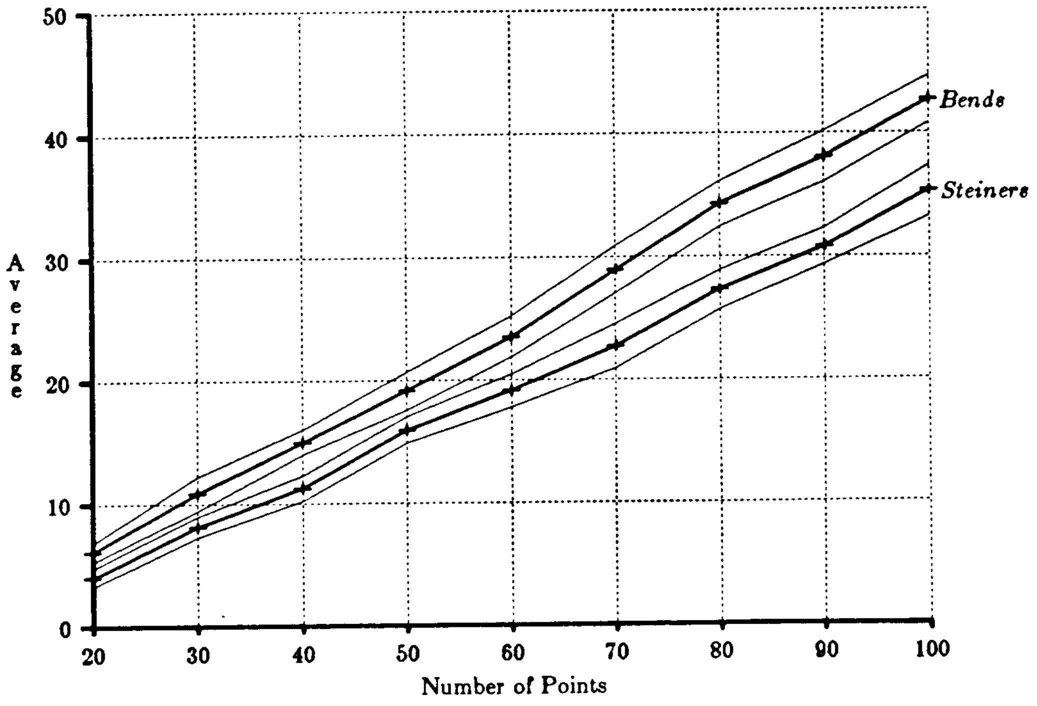


Figure 11. Numbers of bends and Steiner points as a function of problem size.

Reference	Approach	Efficiency	Range of N	% over RMST
Modified Thompson's	Greedy (Kruskal)	$O(N^3)$	20-100	8.9
Lee, Bose, Hwang [LBH]	Greedy (Prim)	$O(N^2)$	5-30	approx. 9
MacGregor-Smith, Liebman [ML]	Rectangularization	$O(N^4)$	5-30	7.2
Yang, Wing [YW]	Branch-and-Bound	expon.	9-50	approx. 11

Figure 12. A comparison of different heuristics.

3.5. Empirical Conclusions

Figure 12 compares our results with other results reported in the literature. The number reported for performance is the mean per cent shorter than a solution to RMST. Overall, Thompson's algorithm produces solutions as short as those found by the other heuristics for RSP, but (neglecting constants) runs more slowly than [LBH] (and related algorithms [H78, H79, MLL], two of which are $O(N \log N)$). It is interesting to note that the one non-greedy polynomial-time heuristic shown [ML] produces relatively poor solutions.

Because it is so difficult to find optimal solutions to RSP (see [L] for two different exponential time algorithms), we do not know how close heuristic solutions are to optimality. Some data on optimal solutions to RSP are included in [Se] and [YW], but the numbers of nodes in the problem instances are too small to permit firm conclusions. If forced to name a number, we would guess that for large problems optimal rectilinear Steiner trees average about 12% shorter than rectilinear minimum spanning trees.

Acknowledgements

We would like to thank Richard Karp for some valuable discussions about the asymptotic analysis in section 2, and Clark Thompson for suggesting this research topic.

References

[AGH] A. V. Aho, M. R. Garey, and F. K. Hwang, "Rectilinear Steiner Trees: Efficient Special Case Algorithms," *Networks* 7, 1977, pp. 37-58.

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[GJ] M. R. Garey and D. S. Johnson, "The Rectilinear Steiner Tree Problem is NP-Complete," *SIAM J. of Appl. Math.*, 32, 1977, pp. 826-834.

[H76] F. K. Hwang, "On Steiner Minimal Trees with Rectilinear Distance," *SIAM J. of Appl. Math.*, 30, 1976, pp. 104-114.

[H78] F. K. Hwang, "The Rectilinear Steiner Problem." *Design Automation and Fault-Tolerant Computing* 2, 1978, pp. 303-310.

[H79] F. K. Hwang, "An $O(N \log N)$ Algorithm for Suboptimal Rectilinear Steiner Trees," *IEEE Trans. on Circuits and Systems* 26, 1979, pp. 75-77.

[HRK] M. Haimovich and A. H. G. Rinnooy Kan, personal communication with A. H. G. Rinnooy Kan.

[KS] R. M. Karp and J. M. Steele, "Probabilistic Analysis of Heuristics", Chapter 6 of *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, edited by E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, Wiley, 1985.

[KoSh] J. Komlós and M. T. Shing, "Probabilistic Partitioning Algorithms for the Rectilinear Steiner Problem," *Networks* 15, 1985, pp. 413-424.

[Kr] J. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," *Proc. AMS* 7, 1956, pp. 48-50.

[L] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, 1976, pp. 289-290.

[LBH] J. H. Lee, N. K. Bose, and F. K. Hwang, "Use of Steiner's Problem in Suboptimal Routing in Rectilinear Metric," *IEEE Trans. on Circuits and Systems* 26, 1976, pp. 470-476.

[ML] J. MacGregor-Smith and J. S. Liebman, "Steiner Trees, Steiner Circuits, and the Interference Problem in Building Design," *Engineering Optimization* 4, 1979, pp. 15-36.

[MLL] J. MacGregor-Smith, D. T. Lee, and J. S. Liebman, "An $O(N \log N)$ Heuristic Algorithm for the Rectilinear Steiner Problem," *Engineering Optimization* 4, 1980, pp. 179-192.

[N] A. Ng, U. of California at Berkeley, personal communication.

[P] H. R. Pitt, *Tauberian Theorems*, Oxford University Press, 1958, pp. 37-42.

[Pr] R. C. Prim, "Shortest Connecting Networks and Some Generalizations," *Bell Syst. Tech. J.* 36, 1957, pp. 1389-1401.

[RT] P. Raghavan and C. D. Thompson, "Provably Good Routing in Graphs: Regular Arrays", *Proceedings of the 17th Annual ACM Symposium on the Theory of Computing*, 1985.

[Se] M. Servit, "Heuristic Algorithms for Rectilinear Steiner Trees," *Digital Processes* 7, 1981, pp. 21-32.

[St] J. M. Steele, "Subadditive Euclidean Functionals and Nonlinear Growth in Geometric Probability," *Ann. Probability* 9, 1982, pp. 365-376.

[T] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, Pennsylvania, 1983.

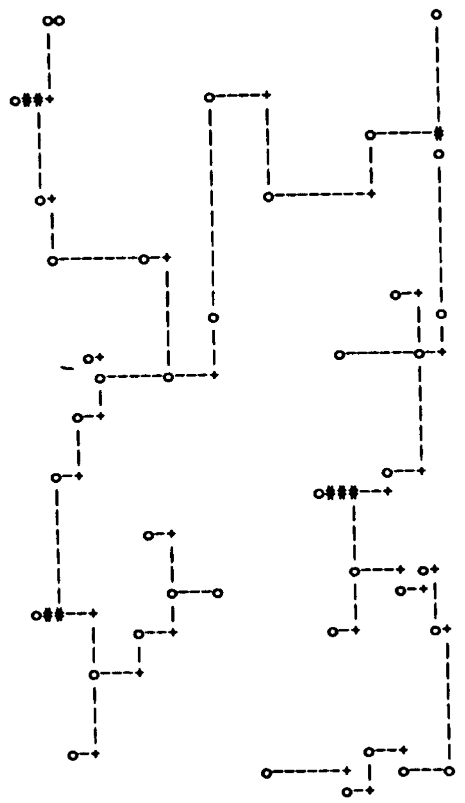
[Th] C. D. Thompson, U. of California at Berkeley, personal communication.

[YW] Y. Y. Yang and O. Wing, "Suboptimal Algorithm for a Wire Routing Problem," *IEEE Trans. Circuit Theory* 19, 1972, pp. 508-511.

APPENDIX A

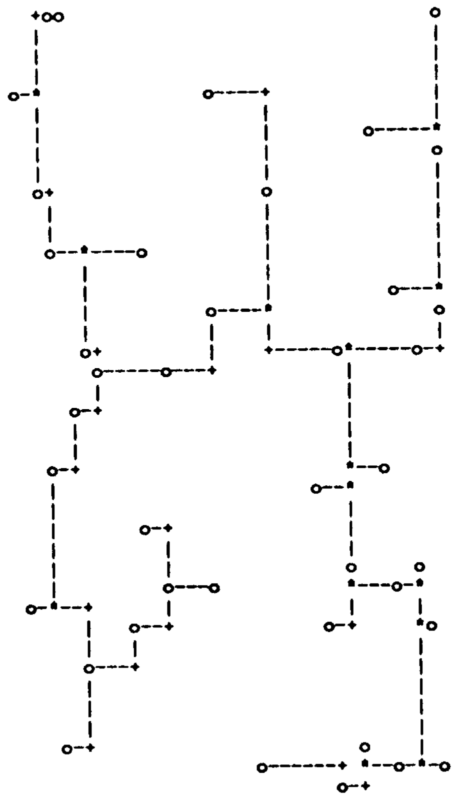
STEINER TREES AND MINIMUM SPANNING TREES

MINIMAL SPANNING TREE



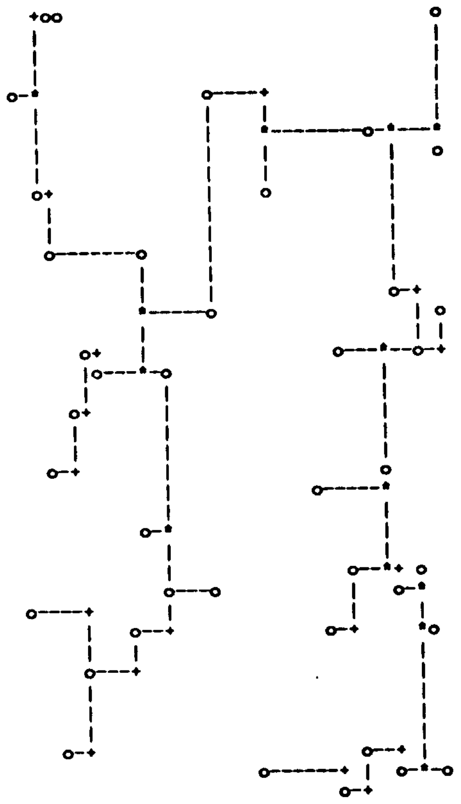
Original points = 40
Double wire = 8
Wire length = 172
Number of Bends = 30
Total length = 202

MODIFIED VERSION



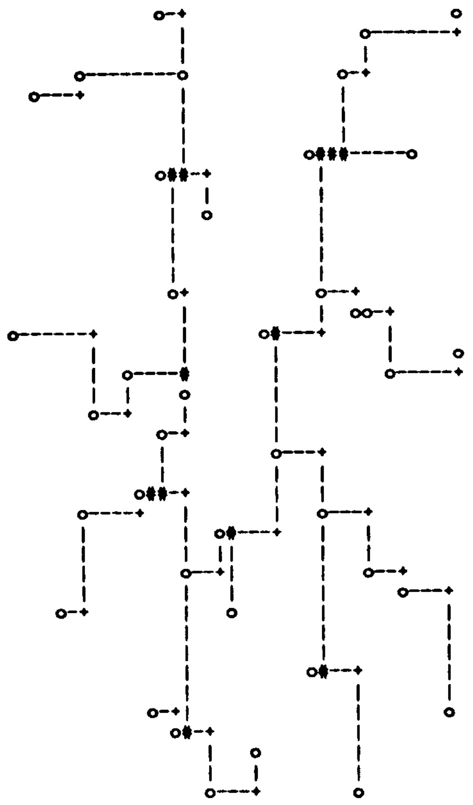
Original points = 40
Steiner points = 14
Wire length = 148
Number of Bends = 17
Total length = 179

ORIGINAL VERSION



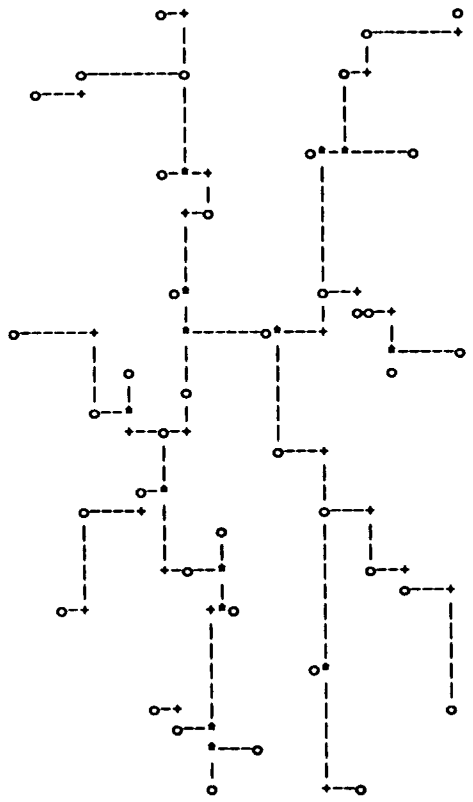
Original points = 40
Steiner points = 13
Wire length = 154
Number of Bends = 17
Total length = 184

MINIMAL SPANNING TREE



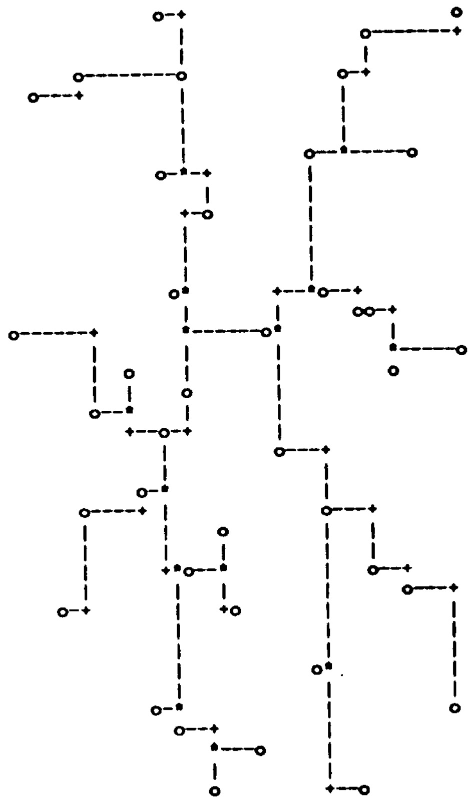
Original points = 40
Double wire = 14
Wire length = 180
Number of Bends = 36
Total length = 216

MODIFIED VERSION



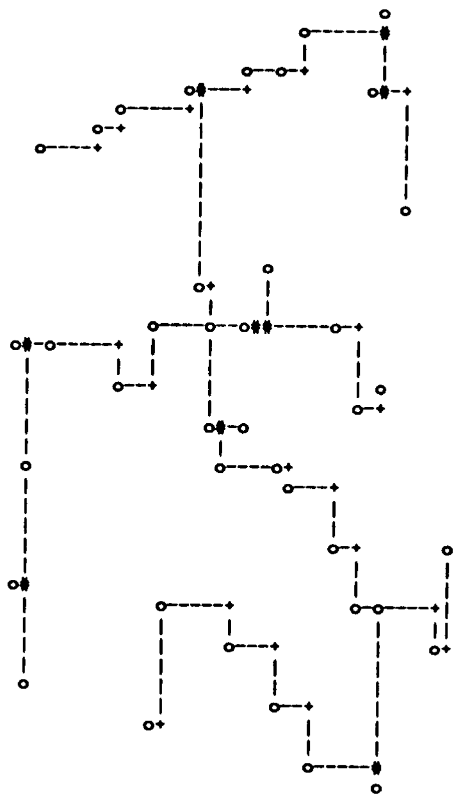
Original points = 40
Steiner points = 14
Wire length = 158
Number of Bends = 22
Total length = 194

ORIGINAL VERSION



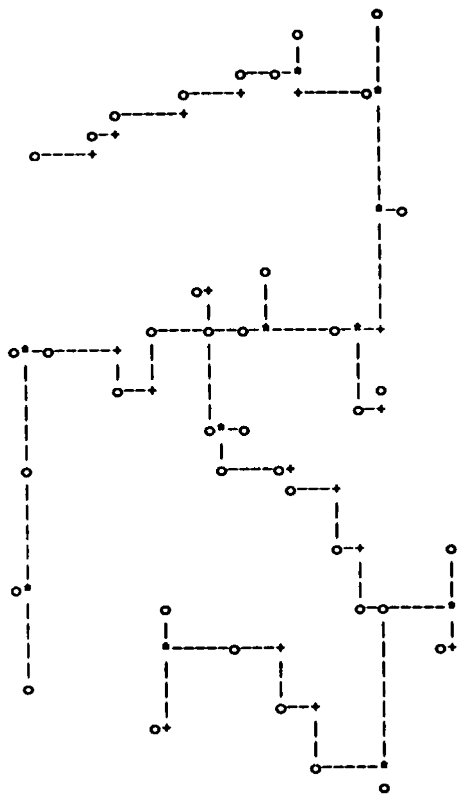
Original points = 40
Steiner points = 14
Wire length = 158
Number of Bends = 22
Total length = 194

MINIMAL SPANNING TREE



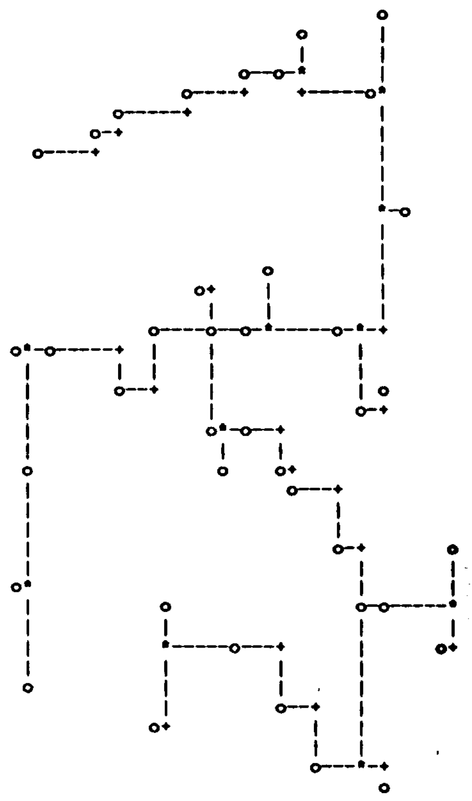
Original points = 40
Double wire = 9
Wire length = 153
Number of Bends = 29
Total length = 182

MODIFIED VERSION



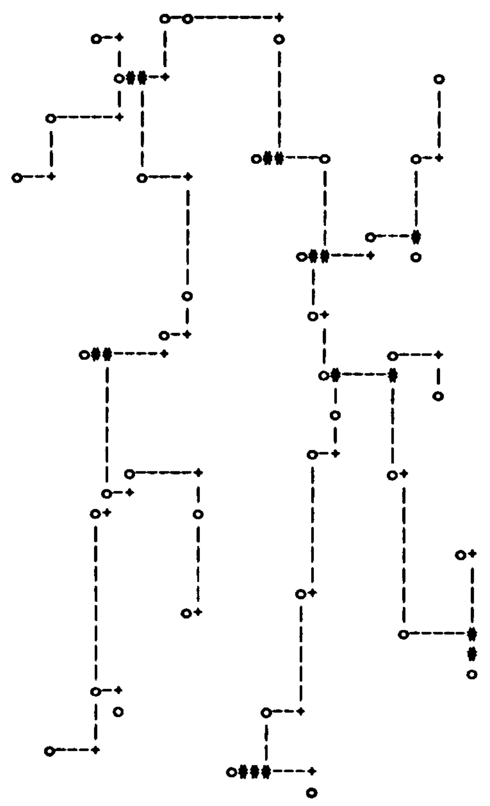
Original points = 40
Steiner points = 11
Wire length = 140
Number of Bends = 17
Total length = 168

ORIGINAL VERSION



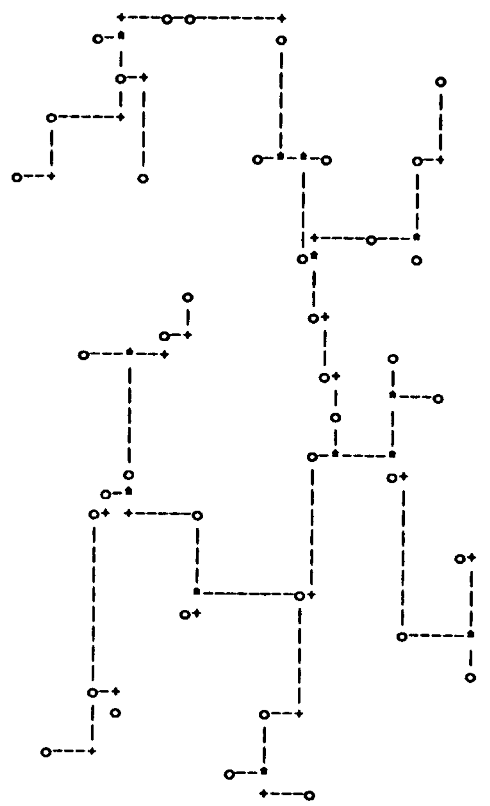
Original points = 40
Steiner points = 11
Wire length = 138
Number of Bends = 19
Total length = 168

MINIMAL SPANNING TREE



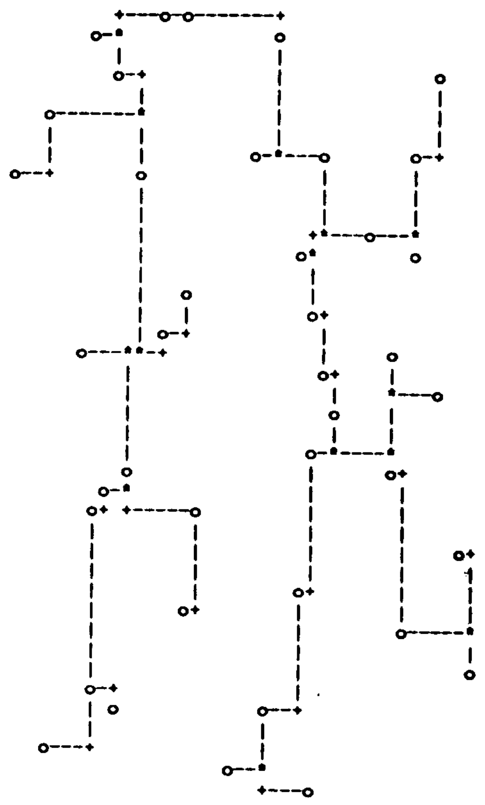
Original points = 40
Double wire = 17
Wire length = 172
Number of Bends = 34
Total length = 206

MODIFIED VERSION



Original points = 40
Steiner points = 13
Wire length = 150
Number of Bends = 21
Total length = 184

ORIGINAL VERSION



Original points = 40
Steiner points = 14
Wire length = 149
Number of Bends = 20
Total length = 183