IMPROVED COMPUTATION OF PLANE STEINER MINIMAL TREES

by

E.J. Cockayne and D.E. Hewgill

DMS–517–IR September 1989

IMPROVED COMPUTATION OF PLANE STEINER MINIMAL TREES

by

E.J. COCKAYNE AND D.E. HEWGILL

Department of Mathematics and Statistics
University of Victoria
P.O. Box 1700
Victoria, B.C. V8W 2Y2
CANADA

ABSTRACT

A *Steiner Minimal Tree* (SMT) for a given set $A = \{a_1, \cdots, a_n\}$ in the plane is a tree which interconnects these points and whose total length *i.e.* the sum of lengths of the branches, is minimum. To achieve the minimum, the tree may contain other points (*Steiner points*) besides $a_1, \cdots, a_n$.

Various improvements are presented to an earlier computer program of the authors for plane SMTs. These changes have radically reduced machine times. The existing program was limited in application to about $n = 30$, while the innovations have facilitated solution of many randomly generated 100-point problems in reasonable processing times.

Keywords: Computation, Steiner Minimal Tree, Plane.

1. Introduction

A *Steiner Minimal Tree* (SMT) for a given set $A = \{a_1, \cdots, a_n\}$ of plane points is a tree which interconnects these points and whose total length, *i.e.* the sum of lengths of the branches, is minimum. To achieve the minimum, the tree may contain other vertices, which are called *Steiner points*, besides $a_1, \cdots, a_n$.

The problem of finding SMTs has been extensively studied. The reader is referred to [1, 13] for background information, applications and bibliographies. We are concerned with so-called "exact" computer programs for SMTs rather than heuristic programs (*e.g.* [12]) which only find suboptimal trees, although the latter may run considerably faster. All exact programs rely on an original geometric idea of Melzak [10]. He showed how to reduce an n-point problem to a set of (n-1)-point problems in a finite number of steps and hence exhibited a geometric (exponential) algorithm for finding SMTs. There is a brief and illuminating description of Melzak's basic idea in [1].

The first computer program for SMTs was written in 1970 (see [6]). Since that date, successive mathematical improvements to the algorithm, innovative programming and of course faster machines have led to more sophisticated algorithms and programs which are practical for higher numbers of given points. In Table 1 we list these methods and their approximate application (using machines available when each was written).

Program	Author(s)	Approximate application
(1) STEINER	Cockayne and Schiller [5]	7 points
(2) STEINER 72	Boyce and Seery [2]	10 points
(3) STEINER 73	Boyce and Seery [3]	12 points
(4) GEOSTEINER	Winter [11]	15 points
(5) EDSTEINER86	Cockayne and Hewgill [5]	30 points (See section 2.2)

Table 1: Existing programs for plane SMTs

We note that the problem is NP–complete [7] and hence it is unlikely that there is any better method than backtracking with efficient pruning of the tree(s) of possibilities.

In [5], the authors described some innovations for Winter's algorithm GEOSTEINER [13]. These were implemented into the program EDSTEINER86 which enabled us to solve all problems of 17 or less given points and an estimated 80% of all problems with at most 30 given points.

In this paper further improvements are introduced. These have been implemented in a new faster program EDSTEINER89 which has successfully handled most randomly generated problems of up to 100 given points.

2. GEOSTEINER and EDSTEINER86

For brevity, we assume that the reader is familiar with Winter's work [13] and the present authors' previous paper [5], which explain any terminology undefined here. A *Full Steiner Tree* (FST) on a plane set B, where $|B| = m$, is a plane tree whose

vertices are the points of B and exactly $m-2$ Steiner points satisfying:

(i) the degree of each point in B is 1, and

(ii) each Steiner point has degree 3 and the angles at each Steiner point are 120°.

It is well known (see *e.g.* [4]) that an SMT for A is an edge–disjoint union of FSTs on subsets of A.

2.1 GEOSTEINER

There have been dramatic increases recently in the size of problems which can be solved, due to an outstanding breakthrough by Winter [13]. His algorithm GEOSTEINER has two principal stages.

Stage 1 is an extremely clever procedure for producing TLIST which is a list of FSTs containing (among others) all those FSTs whose union is an SMT for A. The mathematical pruning techniques for rejecting FSTs from TLIST automatically during processing, are so powerful that randomly generated problems with $n \leq 30$, seem to have at most 100 FSTs in their TLIST and usually considerably less. Furthermore, TLIST for 30–point problems could be generated in no more than five minutes of computing time.

Stage 2 is a standard backtracking procedure along TLIST using a length test, degree test and cycle test, to extract from TLIST the precise set of FSTs whose union is an SMT ([13, Section 6]).

GEOSTEINER's application was restricted to $n \leq 15$ because of stage 2. Winter mentioned that "with further improvements, it is reasonable to assert that point sets with up to 30 V–points (in our terminology, $n \leq 30$) could be solved in less than one hour".

2.2 EDSTEINER86

In this program the authors were able to considerably reduce the one hour estimate in most cases. The first change was the use of an incompatibility matrix for TLIST to assist in speeding up the stage 2 backtracking [5, p. 152].

Due to the exponential nature of SMT procedures, it is highly desirable to invoke theorems of the type: If (A has some geometric property), then the SMT for A is the union of SMTs on subsets $A_1,\cdots,A_k$ (called *blocks*) of A. One such theorem concerns the degeneracy of the Steiner polygon of A and may be applied at the outset *i.e.* before stage 1, (see [5, p. 153]). This block decomposition was included in GEOSTEINER.

Another block decomposition theorem concerning the removal of certain quadrilaterals from the Steiner polygon has recently been proved [9]. It may also be applied before stage 1 and in some cases will facilitate decomposition even when the Steiner polygon is non–degenerate. We have not yet incorporated this theorem into machine programs and mention it here only for completeness. For further discussion of such decompositions in other Steiner tree problems see [11].

Our second improvement was to use the computed TLIST to effect further block decompositions in many cases, ([5, Theorems 2.2 and 2.3]). The blocks induce a partition of TLIST and stage 2 can then be effected on the sets of the partition

separately which, of course, gives vast reductions in backtracking time.

EDSTEINER86 could solve any problem with $n \leq 30$ and maximum block size 17, using no more than 6 minutes of time on the IBM 4381. It is estimated that about 80% of randomly generated 30–point problems in the unit square have this property. For such problems the machine time is dominated by stage 1, while stage 2 used at most 70 seconds. It was found that the time to process stage 2 with EDSTEINER86 may be prohibitive for blocks of size greater than 17 and the present work is devoted to improving this situation.

It should be noted that the Steiner polygon degeneration test is an easy polynomial computation and hence problems with hundreds of given points could be solved by EDSTEINER86 provided that the Steiner polygon test decomposes the problem into blocks of maximum size 17.

3. Algorithmic Changes

In this section, we describe the changes which were made in the construction of EDSTEINER89 from EDSTEINER86. The TLIST for A is now computed by stage 1 and we wish to extract the SMT. For each tree of TLIST, the original points (henceforth called *a–points*), length of the FST, Steiner points and tree topology are stored. It is possible to wait until the SMT has been extracted before constructing any Steiner points [13, p. 332], but Steiner points are needed for our amendments. Also TLIST is usually so short for $n \leq 100$ that the machine time for computing all Steiner points for each tree of TLIST, is insignificant, while the savings facilitated by these points, can be very substantial.

The very simple ideas which have facilitated the surmounting of the 17–point block size barrier, depend on the extension of the concept of compatibility introduced in [5, page 152]. An amended compatibility matrix for TLIST is described in Section 3.1 and in Section 3.2 we indicate that the backtracking for SMT extraction is greatly reduced by performing it on a graph formed from the matrix. Examples of the vast time reductions afforded by these changes appear in Section 4.

3.1 Forest Management of TLIST

Backtracking time can be saved by performing some pre–processing on TLIST, which we call *forest management*. This has two specific purposes: to delete even more FSTs from TLIST and to construct a *compatibility matrix* M for TLIST whose entries are C (meaning compatible), I (incompatible) or D (disjoint). The entry M(i,j) (= M(j,i)) is associated with the unordered pair of FSTs T_i, T_j of TLIST and we wish to construct M so that

$M(i,j) = C$ if $T_i \cup T_j$ could possibly form a (connected) subtree of an SMT.

$= I$ if at most one of T_i, T_j can be in an SMT.

$= D$ otherwise.

As will be seen below, whenever a partial candidate (*i.e.* a set of FSTs) for an SMT has been constructed, the next FST to be added is compatible with some FST of the partial candidate and incompatible with none of them. For efficient, backtracking, therefore, we need to reduce the numbers of C's and increase the number of I's in the matrix M, as far as possible.

The first step in the construction of M, is to make the following assignments:

$M(i,j) = C$ if T_i, T_j have precisely one common a–point and the angle between the branches of T_i and T_j at that point is at least $120°$ (care must be taken with round–off).

$= I$ If branches of T_i, T_j intersect except at an a–point, if T_i, T_j have at least two a–points in common or they have a single common a–point but the angle between these branches at this point is less than $120°$.

$= D$ Otherwise.

In order to describe the refinements to M, we define for each $m = 1, \cdots, n$, the set $F_m = \{T \in TLIST \mid a_m \in T\}$. It is clear that an SMT contains an FST from F_m for each $m = 1, \cdots, n$ and hence some FST from each set F_m must be compatible with any FSTs being considered for inclusion in an SMT. Note that some of the sets F_m will decrease during forest management whenever an FST is deleted from TLIST.

The following refinements are now made to M. T_i, T_j, T_k denote trees of TLIST. The tests (i), (ii) and (iii) are sequentially performed for each $m = 1, \cdots, n$. We assume $j \neq k$.

(i) If $M(j,k) \neq I$, $a_m \notin T_j \cup T_k$ and for each $T_i \in F_m$ at least one of $M(j,i)$ or $M(k,i) = I$,
then change $M(j,k)$ to I.
(No FST from F_m is compatible with both T_j, T_k, hence T_j, T_k are incompatible.)

(ii) If $M(j,k) = C$, $a_m \notin T_j \cup T_k$ and for each $T_i \in F_m$,
either at least one of $M(j,i)$ or $M(k,i) = I$
or $M(j,i) = M(k,i) = C$ and $T_i \cap (T_j \cup T_k)$ has two a–points,
then change $M(j,k)$ to I.
(Each FST of F_m which could possibly be in an SMT with T_j, T_k, in fact forms a cycle with them, hence T_j, T_k are incompatible.

(iii) If $a_m \notin T_j$ and for all $T_i \in F_m$, $M(i,j) = I$,
then delete T_j from TLIST.
(No FST from F_m is compatible with T_j.)

This process of refining the matrix M may be performed several times. Experience has shown that more than two refinements are very seldom required. Machine time for forest management is negligible compared to savings in backtracking time.

3.1.1 Further Refinements in Regular Lattice Problems

We have been consulting with scientists who are concerned with finding SMTs theoretically for certain regular lattice problems *e.g.* $m \times n$ square lattices and some equiangular triangular lattices. The fact that distances between a–points are duplicated in such problems, can often be exploited to delete more trees from TLIST or to show further incompatibilities. We illustrate this situation in Fig. 1 in which:

The a–points a_1, a_2, a_3, are such that the segments a_1a_2 and a_1a_3 have equal lengths and the segment a_1a_2 is also an FST (say T_i) of TLIST.

T_j is an FST of TLIST containing a_3 but not a_1 and is such that $M(j,i) \neq I$ and the angle between the branch of T_j at a_3 and the segment a_3a_1 is less than 120°.

Then the following refinements may be made:

(vi) If T_i is the *only* FST in F_1 such that $M(j,i) \neq I$, then T_j may be deleted from TLIST.

For suppose T_j were in an SMT, say U. Then by hypothesis $T_i \in U$ and a_1 has degree one in U, hence $a_1a_3 \notin U$. Form U′ from U by replacing a_1a_2 with a_1a_3. Since a_1 had degree one in U, U′ is a tree. But U′ has the same length as U and is therefore another SMT. However U′ contradicts the 120° property.

(vii) If T_k is a third FST of TLIST such that $M(k,j) \neq I$ and T_i is the *only* FST of F_1 satisfying $M(k,i) \neq I$ and $M(j,i) \neq I$,
then $M(k,j)$ may be changed to I.

The justification for such a change is very similar to that given for (vi) and is omitted.

3.2 Backtracking

When forest management is completed, the methods of [5] are used for block decompositions (see Section 2.2). Any block A_i has an induced list of FSTs $TLIST_i$ containing all FSTs of TLIST with a–point sets contained in A_i and an induced compatibility matrix M_i which is the submatrix of entries in M, concerning trees in $TLIST_i$. In this section we discuss the backtracking to extract an SMT for a block. To avoid overuse of subscripts, from now on $A = \{a_1, \cdots, a_n\}$, TLIST, M

will denote a block, its induced TLIST and compatibility matrix.

Suppose TLIST = $\{T_1,\cdots,T_t\}$ and for each $i = 1,\cdots,t$, ℓ_i and A_i denote the length and a–points of T_i, respectively. Let G_c be the simple undirected graph with vertex set TLIST and whose edges are the pairs T_iT_j such that $M(i,j) = C$. Each edge T_iT_j is labelled with the index of the a–point which is common to the FSTs T_i and T_j.

We restrict the discussion at this time to randomly generated problems, since in this case, the probability of there being an a–point of degree three in an SMT, is zero. Let X be the set of FSTs in an SMT. Certainly $G_c[X]$, the subgraph of G_c induced by X, is connected. A cycle in $G_c[X]$, with one exception, means a cycle in the assumed SMT. The exception is a triangle in $G_c[X]$ with identical edge labels. This corresponds to the case of a degree three a–point which we have excluded. The restriction also implies that any edge label may only appear once in $G_c[X]$. Any pair of trees in X are compatible and hence the search is for a subset $X = \{T_k | k \in K\}$ of vertices of G_c such that

(i) $G_c[X]$ is a tree.

(ii) $G_c[X]$ has no duplicated edge label.

(iii) For each $k_1,k_2 \in K$, $M(k_1,k_2) \neq I$.

(iv) $\bigcup_{k\in K} A_k = A$.

(v) $\sum_{k\in K} \ell_k$ is minimum.

The solution is now found using a standard backtrack search of the graph G_c. This is far more efficient than the backtracking along the whole of TLIST (without

forest management) which was performed in EDSTEINER86 and GEOSTEINER.

At the jth step, we have a partial SMT candidate *i.e.* a subset X_j of vertices of G_C satisfying (i), (ii) and (iii). At this step the set L_j, disjoint from X_j, of all vertices each of which may possibly be added to X_j to form X_{j+1}, is placed on the top of a stack. Clearly we need $|L_j|$ to be as small as possible. Since each vertex of L_j is compatible with exactly one vertex of X_j and incompatible with none of X_j (so that (i) and (iii) are satisfied in X_{j+1}), the forest management (*i.e.* minimising C's and maximising I's in M) is vital.

Initially $X_0 = \emptyset$ and it is not necessary to place all vertices of X into L_0. It is sufficient that L_0 contains any one of the sets F_{a_i} and we therefore set L_0 to be an F_{a_i} which has minimum cardinality.

The algorithm proceeds by adding the vertex which is on top of the stack to X_j thus forming X_{j+1}. It continues until L_j is empty, the length of the partial SMT candidate exceeds that of the shortest complete SMT candidate found so far or until (iv) is satisfied (*i.e.*we have a new SMT candidate). In each of these situations we backtrack to an earlier step.

Only minor modifications are required to allow a–points to have degree three in an SMT and we do not discuss this further.

4. Computational Experience

EDSTEINER89, which contains about 120 pages of FORTRAN code, was run on a SUN3/60 work station. This machine is approximately five times slower than the main frame

IBM4381 on which EDSTEINER86 was tested.

4.1 Randomly Generated Problems

Two hundred 32–point sets were randomly generated in the unit square. EDSTEINER89 found SMTs for all of these in reasonable machine times and in examples of this size, stage 1 now dominates the total running time. The average time for stage 1 (TLIST construction) was 438 secs and the average time for stage 2 (forest management and SMT extraction) was 43 secs. The average lengths of TLIST before and after forest management were 68 and 60 respectively.

In order to further investigate the effectiveness of the new program and to decide which parts of the procedure most limit this effectiveness, we ran ten randomly generated problems of each point size $10,15,20,25,\cdots,90,95,100$ with a cut–off time of 20 hours. Of these one hundred and ninety problems, only nine were unfinished at the cut–off point, the smallest of these being a 45–point problem.

We now change terminology and define Part 1 of EDSTEINER89 to mean the generation of TLIST and the forest management described in 3.1. Part 2 will mean the SMT extraction from the final TLIST. In all of the one hundred and ninety problems, Part 1 was successfully completed. The average running time for Part 1 and the average length of TLIST do not seem to grow particularly fast with n in randomly generated problems. This fact is suggested by the graph, labelled random, of Fig. 2 in which a quadratic curve $T = 2.02n^2 - 60.9n + 446$ has been fitted to the graph of average Part 1 machine time against n. Obviously sample size is too small for serious deduction.

Finally, we conducted a further test of the effectiveness of EDSTEINER89 for 100–point square randomly generated problems. A hundred of these problems having non–degenerate Steiner polygons were processed, using a cut–off time of 12 hours. Seventy–seven of these were successfully completed. The average machine times (in minutes) were 209 for TLIST construction, 27 for forest management and 10.8 for SMT extraction. Part 1 successfully completed in all of the 100 problems. These times convert approximately to 17.5, 3.5 and 1.1 minutes respectively of mainframe IBM3090 machine time. The average lengths of TLIST before and after forest management were 220 and 200 respectively. Comparison of these numbers with the values for $n = 32$ (above) indicates the slow growth of TLIST length.

Experience has shown that about 15% of these random problems have degenerate Steiner polygons and it is obvious that EDSTEINER89 has a greater success rate on degenerate sets. We therefore suggest that the effectiveness of EDSTEINER89 on square randomly generated 100–point sets is at least 80% with a cut–off time of 12 hours on the SUN3/60. One of the 100–point solutions is depicted in Fig. 3.

4.2 Square Lattice Problems

Regular configurations are often far more difficult to solve than randomly generated sets. The pruning techniques are not so effective and it appears that in order to solve many regular problems, the special geometry must somehow be incorporated into the algorithm. The difficulty is illustrated by the vast amounts of time taken for the following sets A_n of square lattice points. Let $n = 4q + r$ and

$$A_n = \{(i,j) \mid i = 1,\cdots,q \text{ and } j = 1,\cdots,4\}$$
$$\cup \{(1+1,j) \mid j = 1,\cdots,r\}.$$

The machine times in seconds for stage 2 for EDSTEINER86 and EDSTEINER89 for various values of n are compared in Table 2 and a graph plotting the logarithm of the two machine times which emphasizes the exponential nature of the computations, is given in Fig. 4.

The fact that TLIST construction is also far slower for these square lattice problems than for randomly generated problems, is emphasized in Fig. 2. The graph, labelled A_n, shows Part 1 machine times for the A_n problems.

Lastly in Fig. 5 we exhibit SMTs for 45 subsets of $4 \times m$ square lattice points. Each point set is of the form $U \cup V$ where U contains the first k complete columns of the lattice and $k \in \{1,\cdots,5\}$ and V is one of the nine non-empty subsets of the $(k+1)$st column.

n	EDSTEINER86	EDSTEINER89
13	34	7
14	137	14
15	666	43
16	2169	86
17	6419	164
18	26064	552
19	122389	2674
20	395806	6348
21		12555
22		45191
23		437730

Table 2: Stage 2 machine times for the square lattice sets A_n

5. Further Research

The authors are conducting research efforts in the following directions which could lead to further reductions in computer time and hence increase the applicability of the plane SMT program still further.

(i) Introduction of the new quadrilateral removal theorem [9] as mentioned in Section 2.2.

(ii) Further research on the geometry of TLIST. For example, the establishment of block decompositions other than those facilitated by Theorems 2.2 and 2.3 of [5], so that Part 2 computations may be speeded up in cases where these theorems are insufficiently powerful to cause completion within a reasonable time.

(iii) Further refinements to reduce the number of C's and increase the number of I's in the compatibility matrix. For example it may be possible and

useful to dynamically amend the matrix as successive partial SMT candidates are constructed, *i.e.* the fact that certain FSTs are to be included could obviously change the compatibility of other pairs of FSTs.

(iv) There is now a linear time algorithm which will construct (if possible) an FST given a topology [8]. F. Hwang and P. Winter (private communications) have suggested that the ideas involved in that algorithm could be used to considerably improve Part 1.

(v) Use of the special geometry to increase the applicability of the program for plane square lattice point sets .

Acknowledgements

The authors gratefully acknowledge the support of the Canadian Natural Sciences and Engineering Council Grant #A–7544 and A–7558.

References

[1] M.W. Bern and R.L. Graham. The Shortest Network Problem, *Scientific American*, Jan. 1989, 84–89.

[2] W.M. Boyce and J.B. Seery. STEINER 72 – An improved version of Cockayne and Schiller's program STEINER for the minimal network problem, Tech. Rept. #35, Bell Labs., Dept. of Computer Science, 1975.

[3] W.M. Boyce and J.B. Seery. STEINER 73, Private communication.

[4] E.J. Cockayne, On the efficiency of the algorithm for Steiner Minimal Trees, *SIAM J. Appl. Math.*, **18**(1), (1970), 150–159.

[5] E.J. Cockayne and D.E. Hewgill, Exact computation of Steiner Minimal Trees in the Plane, *Inf. Proc. Letters*, **22**(1986), 151–156.

[6] E.J. Cockayne and D.G. Schiller, Computation of Steiner Minimal Trees, in: D.J.A. Welsh and D.R. Woodall [eds.] *Combinatorics* (Inst. Math. Appl. 1972), 52–71.

[7] M.R. Garey, R.L. Graham and D.S. Johnson. The complexity of computing Steiner Minimal Trees, *SIAM J. Appl. Math.*, **32**(4), (1977), 835–859.

[8] F. Hwang. A linear time algorithm for full Steiner trees, *Op. Res. Letters*, 4(5), (1986), 235–237.

[9] F.K. Hwang, G.D. Song, G.Y. Ting and D.Z. Du. A decomposition theorem on Euclidean Steiner Minimal Trees, *Discrete Comput. Geom.*, **3**(1988), 367–382.

[10] Z.A. Melzak. On the problem of Steiner, *Can. Math. Bull.*, 4(1961), 143–148.

[11] J.S. Provan. The role of Steiner Hulls in the solution to Steiner Tree problems (submitted).

[12] 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(1981), 23–29.

[13] P. Winter. An algorithm for the Steiner problem in the Euclidean plane, *Networks*, **15**(1985), 323–345.

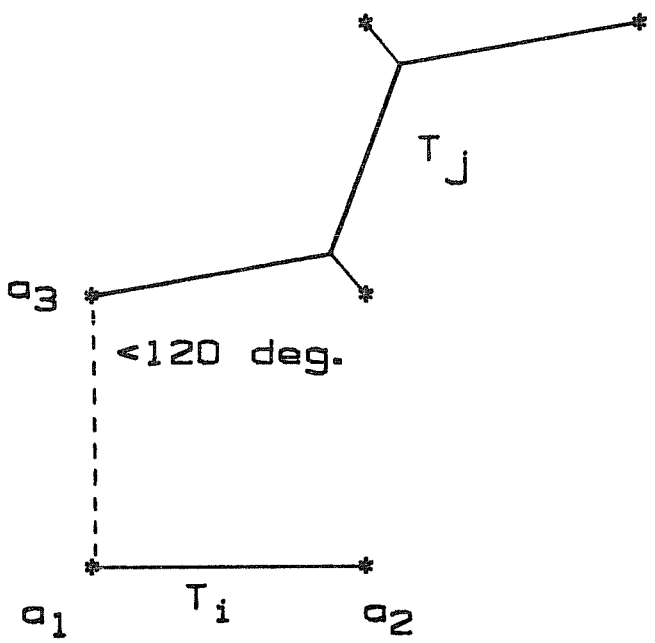

Fig. 1 Forest management with duplicated distances

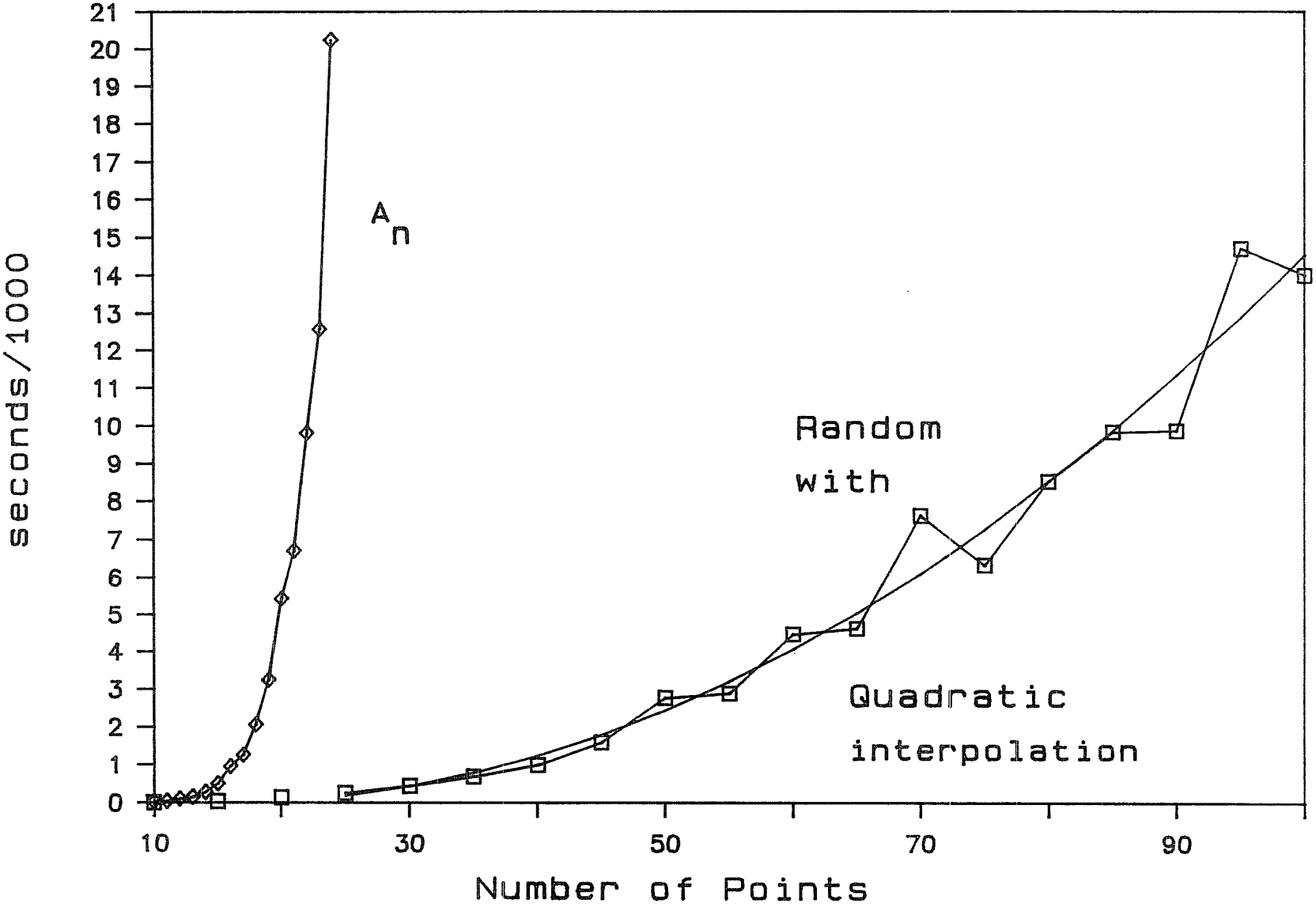

Fig.2 Part One computer time

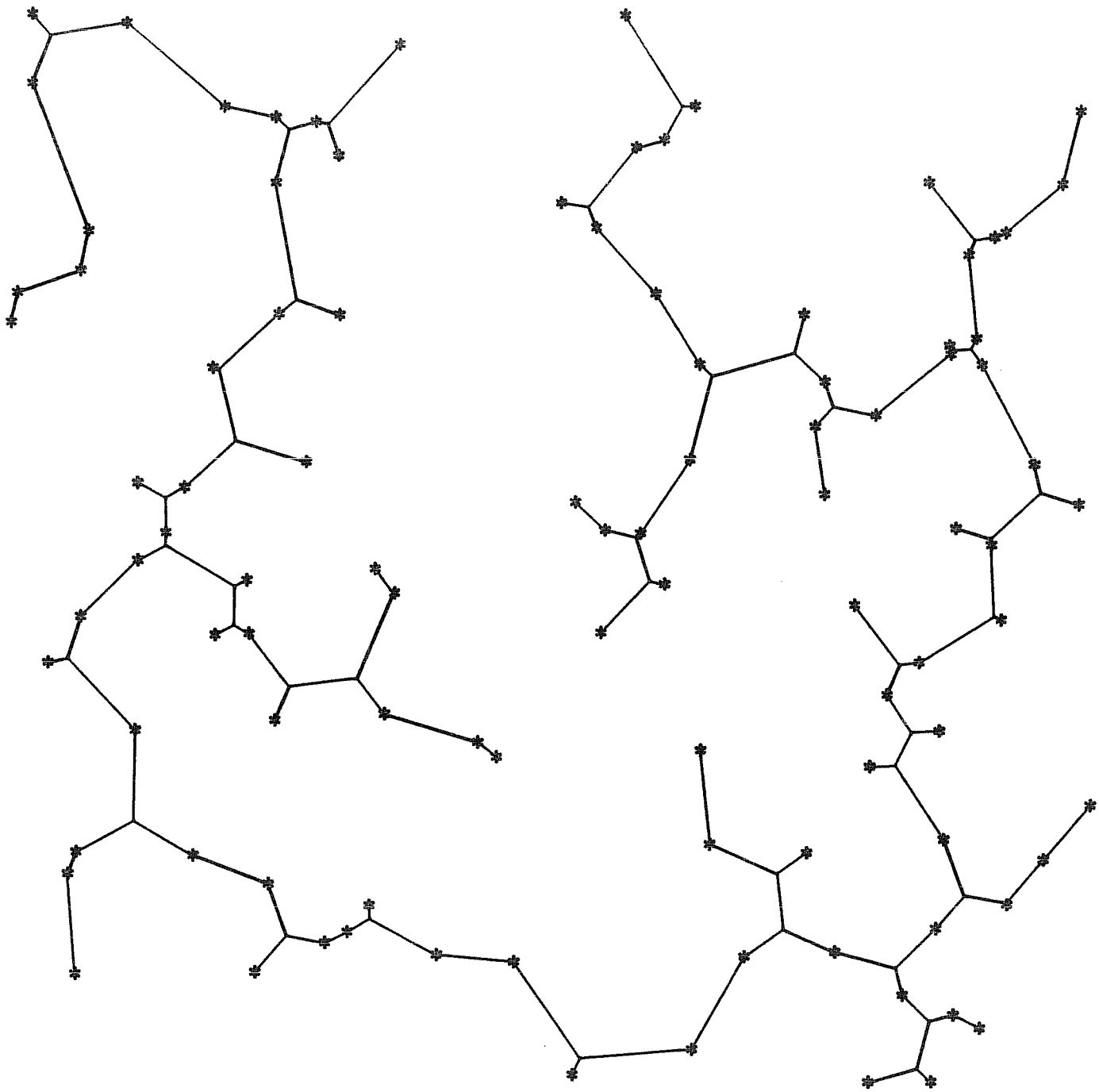

Fig. 3 A 100 Point example (random)

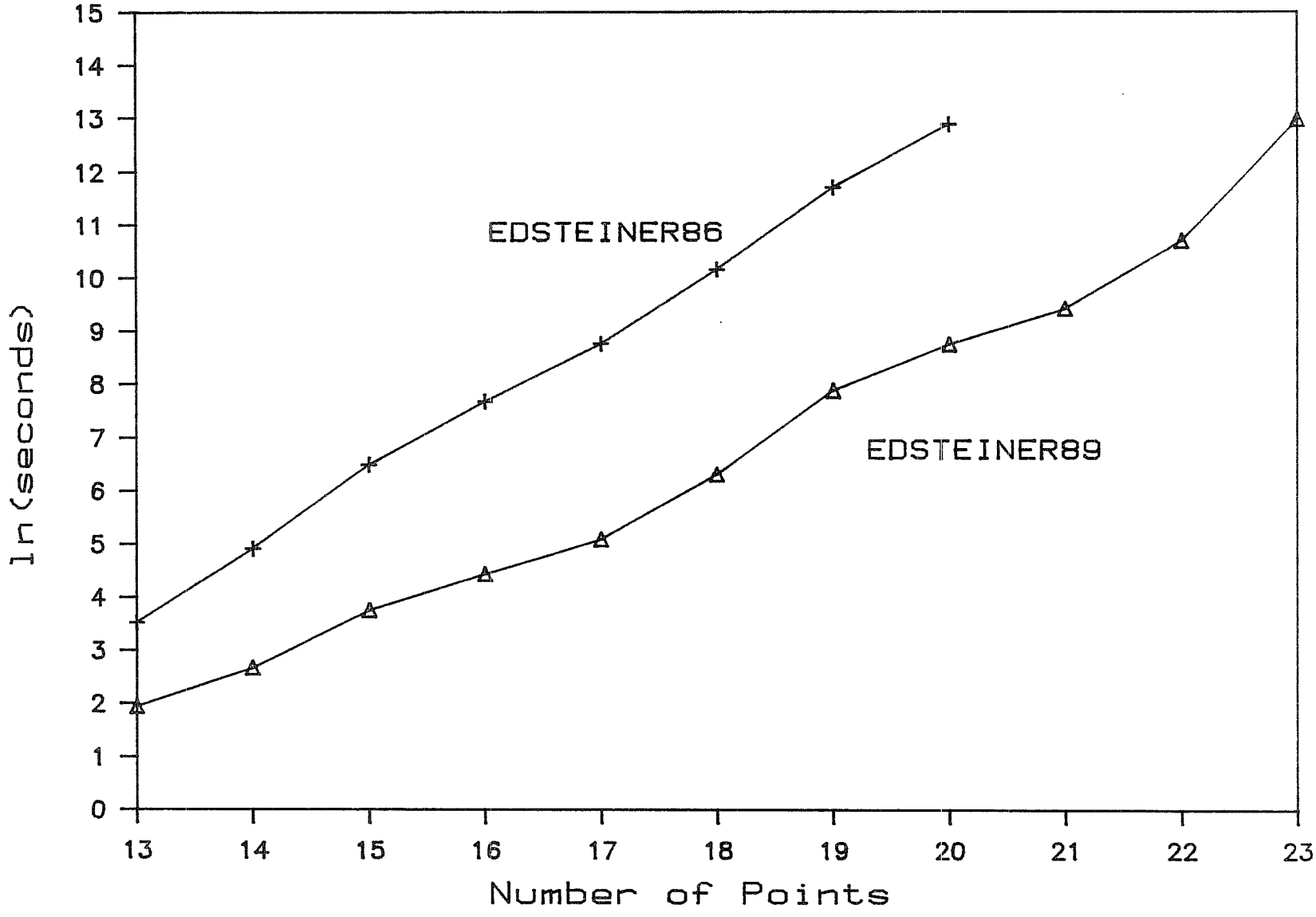

Fig. 4 Logarithms of stage 2 Computer time for A_n

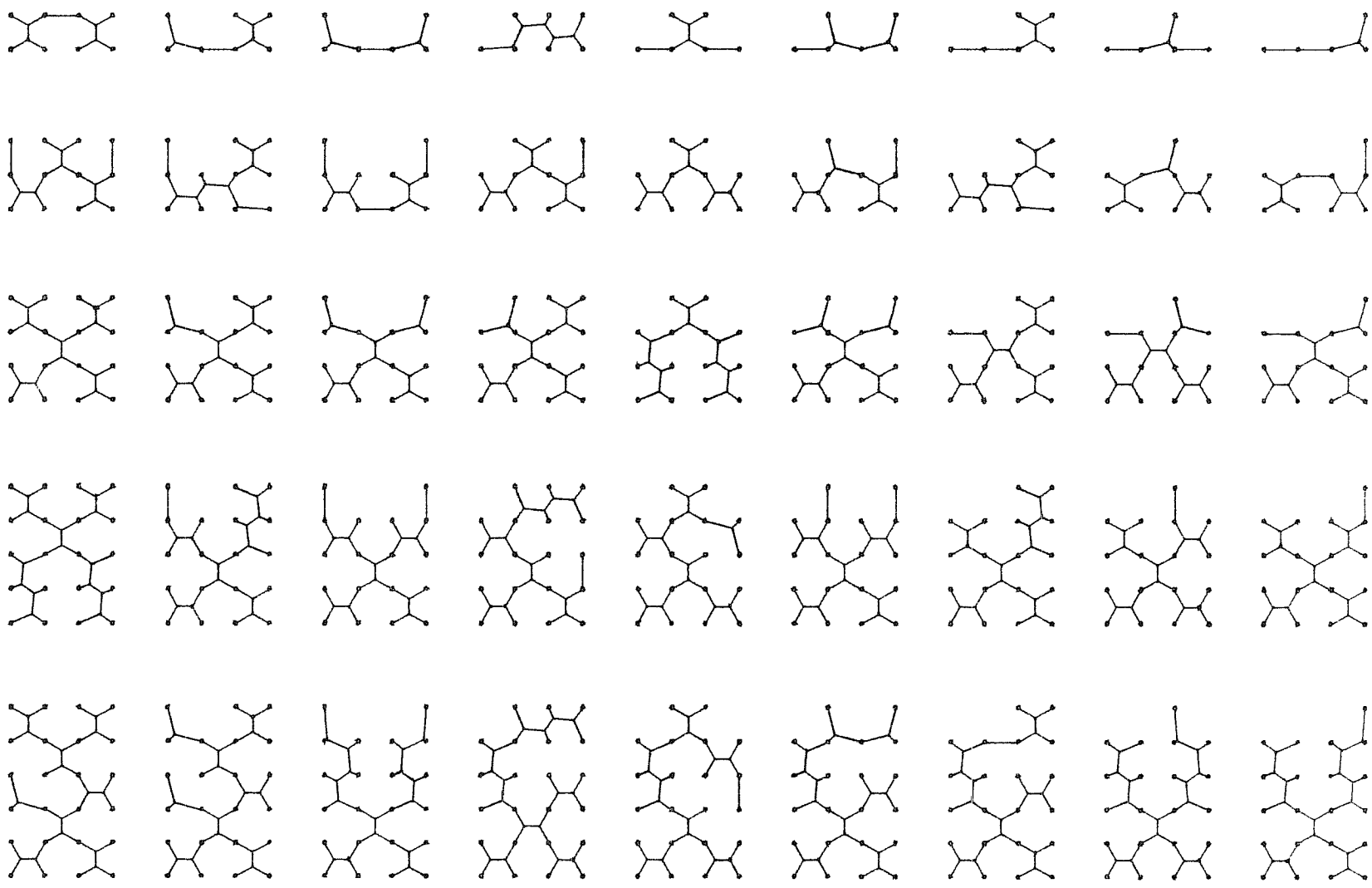

Fig.5 SMT for subsets of the 4 by n square lattice