20

THE PLANNING APPROACH TO INTERACTIVE PROBLEM SOLVING AND THE TRAVELLING SALESMAN PROBLEM

by

William E. Howden

TECHNICAL REPORT #20 - AUGUST 1972

CONTENTS

1. Introduction

The purpose of this report is to describe an interactive problem solving system based on the ideas of planning developed and explained in (1). In particular, the system can be used to interactively construct solutions to Euclidian travelling salesman problems. The goals of the research described here were: (i) to examine how the notions of planning in (1) could be used to construct a system in which the user can have and try out general or abstract ideas for a solution and (ii) to assess the value of the approach as a method for reducing the combinatorial computation requirements for the travelling salesman problem by allowing the user to direct or plan the computational activities of the machine.

The report begins by describing the internal data structure of the system. This consists of a tree of subproblems which the user constructs through the use of a display and a RAND Tablet Pen. "Bottom level" subproblems in the tree are sub-TSP problems which contain subsets of the original set of cities. "Intermediate and higher level" subproblems are abstract travelling salesman problems for which the"cities" are transformations of lower level subproblems into a new space of cities.

The subproblem solvers are then described. The user operates the system by creating, deleting and solving subproblems. At any time he can request that some created subproblem be solved by one of the subproblem solvers. He can also cause the "synthesis" of all the subproblem solutions in some tree of subproblems through the use of a synthesis command.

Because the system is used for Euclidian problems, subproblem solutions, synthesized solutions, subproblem definitions, and subproblem and city names can all be conveniently displayed and referenced with a graphics console provided with a RAND Tablet. The next section of the report describes some of the commands available to the user. Interesting features of some of the command algorithms are described in some detail.

The following sections contain protocols or "traces" of actual problem solving sessions for three different problems. Replicas of the displays which existed during the solution process are included to help give a clear description of the interplay between the users solution ideas, his ability to express them to the machine in the system, and the corresponding action taken by the machine.

The report concludes with a summary of our experience with the system. We discuss how the system achieves the goals mentioned above and some of the system limitations which were discovered.

Appendices contain documented listings of the system programs.

2. Internal Data Structure

Recall that in the R-Plan formalism for subproblem or planning problem solving, solution structure consists of a tree of subproblem "non-terminals". Each non-terminal has an intentional and an extensional definition as well as certain computed properties. Intentional definitions are descriptions of subproblems whose solutions are "pieces" of graph-theoretic or combinatorial representations of solutions. Extensional definitions are subproblem solutions and consist of two parts: (i) a list of the "objects" (terminals or non-terminals in the R-Plan) in the solution and (ii) a structural description of the relationships between the objects which bind them into graph-theoretic or combinatorial solution pieces. For a high level non-terminal, the structure binding the objects in its extension will be a high level description of certain aspects of the solution to the whole problem.

The structural part of an extension can be extended to include "global" as well as "local" structure. Global structure relates any of the objects in the subtree rooted at a non-terminal and not just those in the extension of

that non-terminal. (E.g.it might relate the objects in the extensions of the objects in the extension of the non-terminal, and so on.)

The (computed) properties of a non-terminal will probably include a name or label for the non-terminal. In a standard phrase structure grammar, the name of a non-terminal is also its intentional definition (e.g.verb). For both picture language and problem solving non-terminal structures, it is important that each non-terminal have a unique name or label as well as its intentional definition.

The internal data structure for the interactive TSP system is modelled very closely after the general R-Plan subproblem structure. We shall assume a familiarity with the R-Plan notions in describing the TSP system.

(a) Intentional Definitions. Subproblem specifications in this system consist of two parts: polygons and subproblem end points or boundary points.

In describing (creating) a subproblem in the system the user draws a polygon on the display screen and suggests two end points. The subproblem is then to construct a path from one endpoint to the other which passes through all of the cities in the enclosing polygon. If the polygon contains another polygon, then that polygon (non-terminal) is treated as a "super-city" situated at the centroid of the centroids of its enclosed cities or non-terminals. The centroid of a city (terminal) is just its coordinates in the plane. A city is enclosed in a polygon if its coordinates are geometrically inside the polygon and if it is not enclosed in some other polygon which is itself enclosed inside the original polygon.

Note that non-terminal intentions describe <u>linear</u> TSP's and not <u>circular</u> TSP's. A linear TSP consists of the request to find a shortest path through a set of cities (perhaps containing some "super cities") which starts at one given endpoint and ends at another. If the two endpoints are the same city

then we have a circular TSP. Except at the top level, all subproblem solutions will be linear and will hence be defined by both polygons and endpoint pairs. (The endpoints are called subproblem boundary points because they are the parts of a subsolution which can be joined to other subsolution endpoints in order to synthesize an entire solution.)

We note at this point that the solution to subproblems cannot be accomplished by constructing a circular tour which is then broken in the most appropriate place to join it with other subtours. Consider the following example:

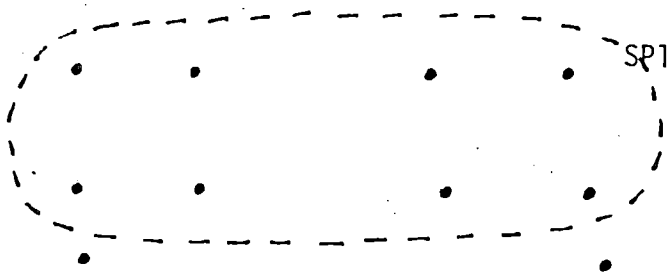


Figure 1.

Suppose a circular optimal tour is constructed for SP1 (subproblem 1) (Figure 2).

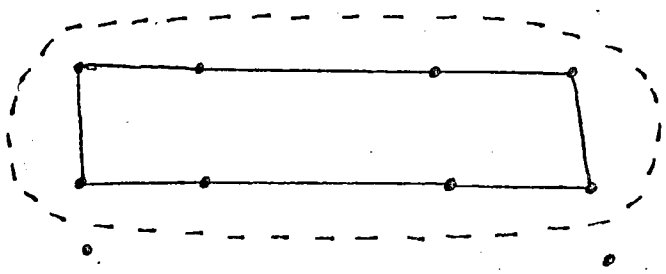


Figure 2.

There is no single cut in SP1 which can be used to merge the SP1 solution with other subproblem solutions in order to form the optimal total problem solution (Figure 3).

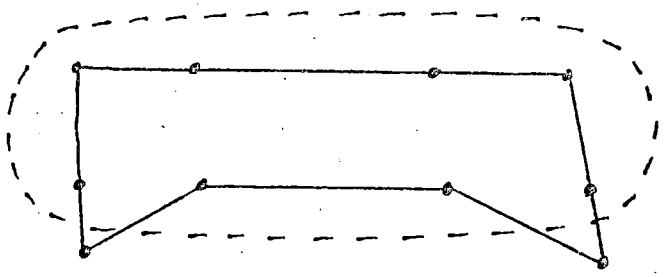


Figure 3.

In forming ideas for subproblems the user is expected to contribute both the polygon and the endpoints. The polygon results from his having recognized a certain pattern in the problem. The endpoints choice results from his understanding of the context of the subproblem: the way in which this subproblem will be merged or joined with other subproblems and their solutions. We note that it is possible to imagine a system in which subproblems endpoints are chosen automatically by the system according to some heuristic. Experience has shown, however, that the obvious choice for such a heuristic does not work and that choosing alternative subproblem endpoints is as important to the user in suggesting problem ideas as choosing enclosing polygons.

The two different parts of non-terminal intention (i.e. subproblem definition) can be defined at different times during the solution process. In using the system it is customary to define the enclosing polygons at one point, and to only name the subproblem endpoints when asking for a subproblem solution to the subproblem associated with some polygon.

(b) Extensional Definitions. Since each terminal and non-terminal in the subproblem tree has a unique name a set of objects (terminals or non-terminals) can be denoted by a list of the object names. The names of terminals are city names and the names of non-terminals are subproblem names.

The structural part of a non-terminal can also be represented by a list of object names. Since a solution to a TSP problem or subproblem is an ordering of objects, it can be represented by a list of the object names written in the order of the solution. For linear tours or solutions, each object's name appears exactly once in this list. For circular tours each object's name appears exactly once except for the first object name which will be the same as the last object name. Local structural descriptions will be a list of the objects in some non-terminal's extension. In this system global structural descriptions are synthesized solutions. For a given tree of non-terminals/terminals rooted at some non-terminal, a global structural description will be the largest solution synthesis which can be formed at that point in the solution process. If every subproblem (non-terminal) in the tree has been solved, the solution synthesis will be a list of cities. If some non-terminal has not been solved, it will appear in the global structural list at the appropriate point and will be the "representative" for all the cities at its terminal nodes, whose local ordering has yet to be determined.

(c) Object Properties. Every terminal and non-terminal has a name, a centroid, a terminal property, and a belongs property. The terminal property of an object is a flag which signifies whether or not the object is a terminal or a non-terminal. The belongs property is the name of the object whose extension includes this object.

Every object "belongs" to exactly one other object except the top level object whose name is UNIV. This implies that extensions cannot intersect. Our experience with the system does not indicate that it is necessary to become involved in the difficult problems that would have to be solved in the construction of a system which allowed overlapping extensions. The design of the command structure (see the description of the CSUB command) is

such that overlaps cannot be constructed - deliberately or otherwise.

Initially, the subproblem, or solution phrase tree, consists of a simple tree of one non-terminal node UNIV and as many terminal nodes as there are cities. Initially then, each terminal belongs to UNIV. As subproblems are created and destroyed, the altered extension memberships are automatically altered and updated by the system. When an action has side effects (e.g. the deletion of a subproblem from an extension will make the structural part of that non-terminal - as well as the list of objects - invalid) these are also processed automatically.

(d) LISP Data Structure. The internal data structure is implemented using LISP atoms, properties and lists. Each terminal or non-terminal is a LISP atom. Intentional definitions, extensional definitions, non-terminal properties and labels are all properties of atoms. The name of an object is the print name of the associated atom. The LISP system proved to be very well suited to the implementation of our interactive system.

(e) Solution Status. In addition to the subproblem data structure and various system tables (e.g. the inter-city distance matrix) a status word is kept. The idea of a status word is to explicitly record pertinent information about the state of the solution process which may be awkward or impossible to retrieve from the subproblem data structure. In the present system it was necessary to store only one status item called the context. The context is the name of the present non-terminal which is considered to be the problem solving universe. Initially the context is UNIV and subproblems can be created from any of the objects in the extension of UNIV (thus automatically changing this extension). If the user wishes to create a subproblem (non-terminal) which is to be a subproblem of some other non-terminal than UNIV, then he must change the context to that other non-terminal. At present this is the only "context operation". No other operations depend on the present context. The feature is necessary for subproblem creation in order

to determine which subproblem in a nest of subproblems is being referred to by an enclosing polygon: the subproblem in the extension of the present context.

Other useful status information would be records of the state of the present display picture. Since the display files and data structures are separate, and it is not possible to examine the display file to determine the present picture, this would be helpful when features of the display are changed automatically by changes in the data structure . Since this feature was not necessary for the purposes of our experiments, it and other status features were not implemented.

3. Display File Structure

We have mentioned above that the display file structure and the problem solving data structure are separate. The display file structure is stored and manipulated by the display mini-computer operating system. The problem data structure is created in the PDP-10 LISP system. A number of display commands will change the display file structure. These commands can be issued directly by the user or can be issued automatically by certain problem solving commands.

The display information is organized on different frames. Different aspects of the display can therefore be erased or displayed without altering others. For example, one frame is used to display cities, another subproblem polygons, another subproblem solutions, another subproblem names, and so on. The features of the display structure and its interface with the internal data structure result from the straightforward use of the IMLAC IMSYS monitor (2) and the LISP graphics package (3).

4. Subproblem Solvers

The interactive system has been characterized in terms of the user

manipulation of plans through the creation, solution, alteration, etc. of subproblems. There are presently three subproblem solving processes in the system which may be invoked by the user. They can be graded in terms of their power and their cost. The first is a non-heuristic procedure which produces guaranteed optimal solutions but which is therefore very expensive to use. The second is a very powerful heuristic procedure which is considerably cheaper to use. The third is a relatively weak but very cheap heuristic procedure.

5. The Karp Dynamic Programming Subproblem Solver.

The non-heuristic subproblem solver is based on a slight modification of Held and Karp's Dynamic Programming approach described in (4). The modifications are those which were required to use the method for subproblem solving (i.e. construct linear solutions for linear problems) as well as for conventional TSP problem solving (i.e. produce circular tours).

The approach is attractive since it compares well with other non-heuristic procedures (5) and its computation time and storage requirements are deterministic. The user knows exactly what to expect from it in an interactive situation. Experience shows that it can be conveniently used for up to 12-15 cities in an interactive environment. The computation time (and possibly storage requirements) are prohibitive for larger problems. For technical reasons (INUM and FIXNUM sizes in the PDP-10 LISP) the maximum number of cities the solver will handle in our system is 12.

(a) Recursive Formulation. The interpretation of the method implemented in our system is based on the following recursive formulation of the linear tour travelling salesman problem.

Assume that we wish to go from city 1 to city n. Then the length of the minimum path, MINP , is given by (i) and (ii).

$$\text{(i)}\quad MINP = \min_{h \in \{2,3,\ldots,n-1\}} \{F(\{2,3,\ldots,n-1\},h)+c_{h,n}\}$$

where $c_{h,n}$ is the cost of the path from city h directly to city n. In the Euclidian TSP, $c_{i,j}$ is simply the Euclidian distance from city i to city j.

$$\text{(ii)}\quad F(S,h) = \begin{cases} c_{1,h} & \text{for } S=\{h\} \\ \min\limits_{i \in S \sim h} \{F(S \sim h,i)+c_{i,h}\} & \text{otherwise} \end{cases}$$

F(S,h) is the minimum path from city 1, through the cities in S and ending at city h.

A simple recursive implementation of this recursive representation would be grossly inefficient. A basic idea in the dynamic programming approach is to avoid recomputing previously computed information during the recursive process. Since it is possible to arrive at the computation of many F(S,h)'s along different paths in the recursion process, these values must be saved for possible future reference. Karp's two phase approach was adopted in which the values of F(S,h) were first computed for all possible choices of (S,h) and then used to construct the optimal path. Whenever a value for some F(S,h) is required during the computation process in the first stage, the table of presently computed values is first checked before beginning a new recursion on that F(S,h). A simple recursive process in which values were not saved or referenced in phase one would require on the order of (n-2)! computations of F. When values are saved and referenced the number is of the order $(n-1)(n-2)2^{n-3}$.

Once the table of values of F(S,h) has been computed, the following formulae can be used to construct the optimal path. The construction proceeds in typical Dynamic Programming fashion: by working backwards from the last node in the path.

Let C be the path length of the optimal path. Then a permutation $(1, i_2, i_3, \ldots, i_{n-1}, n)$ is optimal (i.e, represents the optimal path) if and only if:

(iii) $C = F(\{i_2, \ldots, i_{n-1}\}, i_{n-1}) + c_{i_{n-1}, n}$

and for $2 \leq p \leq n-2$,

(iv) $F(\{i_2, i_3, \ldots i_p, i_{p+1}\}, i_{p+1})$

$= F(\{i_2, i_3, \ldots, i_p\}, i_p) + c_{i_p, i_{p+1}}$.

First i_{n-1} is determined, then i_{n-2}, and so on.

(b) Storage of Intermediate Values of F. A limiting factor in the application of the algorithm is the storage required for the saved values of F(S,h). Since there are $\binom{n}{k}$ ways of selecting an S with k elements and k ways of choosing h from S, there are

$$\sum_{k=1}^{n-2} \binom{n-2}{k} k = (n-2) \sum_{k=0}^{n-3} \binom{n-3}{k}$$

$= (n-2)\, 2^{n-3}$ by the binomial theorem.

different possible choices of (S,h). (We eliminate the choice of either the first or last elements (i.e. the endpoint cities) for S since these are in all solution permutations in the same places).

In the first phase of the solution process it is necessary that the storage and lookup of values of F(S,h) be reasonably efficient. For this reason an indexing scheme for the different possible choices of (S,h) was devised to allow the values of F(S,h) to be stored in a simple linear vector.

The scheme for indexing the (S,h) depends on the canonical representations for the (S,h)'s. The canonical representation of an (S,h) is constructed by listing the elements of S in order of magnitude (the use of this procedure requires the city names be mapped 1 to 1 into a list of consecutive integers)

and then underlining the element h.

Eg. The canonical representation for (S,h)

= ({5426} ,4) is 2456.

The pairs (S,h) (and hence the values for the F(S,h)'s) are indexed first on the basis of the cardinality of S, then on the lexicographical ordering of all sets S of that cardinality, and then on the position of the underlined element in the canonical representation of (S,h). Before describing the process in detail we will give an example. Suppose the number of cities n in the problem is 6. Then if we store all (S,h)'s for S with cardinality between 2 and 4, the indexing process will order the values of F(S,h) in the table according to the ordering of the cannonical representations of all choices of (S,h)'s in figure 4.

23	234
23	235
24	235
24	235
25	245
25	245
34	245
34	345
35	345
35	345
45	2345
45	2345
234	2345
234	2345

Figure 4.

The size of the table when values of $F(S,h)$ are not stored for singleton sets S is

$$\sum_{i=2}^{n-2} \binom{n-2}{i} i = \sum_{i=1}^{n-2} \binom{n-2}{i} i - \binom{n-2}{1}$$

$= (n-2)2^{n-3} - (n-2)$ by the binomial theorem.

The indexing function for the table will be described in three parts.

(i) Level Factor. The entries for all (S,h)'s with LEV elements begins at position

$$A = \sum_{i=2}^{LEV-1} \binom{n-2}{i} i$$ where n is the total number of cities.

(ii) Selection Factor. Within a level (the set of all entries for (S,h)'s for which S has a constant cardinality) the entries are ordered lexicographically. Suppose d_i, i=1,2,...,LEV,are the city numbers in S listed in canonical order. If we define $d_0 = 1$, then the entries for the pair (S,h) where S consists of the LEV city numbers $d_1, d_2, \ldots, d_{LEV}$ will begin at $A + B$ where

$$B = LEV \sum_{i=1}^{LEV} \sum_{j=d_{i-1}+1}^{d_i-1} \binom{n-j-1}{LEV-i}.$$

(iii) Fixed City Factor. If p is the position of the underlined city, then the exact position of (S,h) where S has the LEV city numbers $d_1, d_2, \ldots, d_{LEV}$ is

$$A + B + p.$$

In the CKSOL (create a Karp Solution) subproblem solution program, the indexing process is optimized through the use of a table of binomial coefficients and by only computing the different parts of the indexing process as they are needed.

(c) Computation Time. Since the algorithm revolves around the storing and reading of values into and from the table of values for F(S,h) we would expect the computation time required to be related to the size of this table. The basic operations required are additions and comparisons. The number of each in the first phase is given by:

$$(n-1) + \sum_{k=2}^{n-1} k(k-1) \binom{n-1}{k} = (n-1)(n-2)\,2^{n-3} + (n-1).$$

The number of occurrences of each in the second phase is at most

$$\sum_{k=2}^{n-1} k = \frac{n(n+1)}{2} - 1.$$

Experience has indicated that our program requires less than a second for a six city problem and 8 or 9 (CPU) seconds for a ten city problem. Because of the deterministic nature of the algorithm its computation time will not vary from problem to problem.

Shen Lin has described a technique for speeding up Karp's algorithm for symmetric TSP's (4). Unfortunately the technique cannot be applied to the linear TSP algorithm. For subproblems with n cities, Lin's algorithm can be used to produce solutions in (for n even)

$$\sum_{k=2}^{n/2} \binom{n-1}{k} k \quad \text{computational cycles}$$

whereas Karp's algorithm will operate in

$$\sum_{k=2}^{n-2} \binom{n-2}{k} k \quad \text{cycles. It is easy to calculate that for}$$

n in the range of interest,

$$\sum_{k=2}^{n-2} \binom{n-2}{k} k \quad < \quad \sum_{k=2}^{n/2} \binom{n-1}{k} k.$$

Other less elegant improvements (including bit-picking programming) can be used to speed up our program but for our purposes it is not worth the bother. The important parts of the algorithm have been carefully coded and the decrease in computation time would not be important. Since computation time for the program grows exponentially, more clever coding is unlikely to raise by more than one or two the number of cities which can be dealt with within the response times required in an interactive environment. Experiments with a system in which the "exact" subproblem solver can deal with several more cities are unlikely to suggest significantly different conclusions.

6. The Lin and Croes heuristic subproblem solvers.

The Lin and Croes procedures are "hill-climbing" algorithms. In the hill-climbing approach a random initial solution is chosen and then successively modified until a "better" solution is found. The algorithm is then restarted at the better solution. When all possible modifications from a fixed set of modifications have been tried, and no further improvement in the solution can be accomplished, then a locally optimal solution has been discovered. The modified solutions which can be produced from a solution are called its neighbours. If the procedure is applied to several different randomly chosen initial solutions then the final solution for the problem is the best locally optimal solution.

Two different hill-climbing approaches can be identified. The first, called steepest ascent (descent) requires that the best neighbour be chosen at each step of the solution process. In this approach all of the neighbours of a solution are considered before the procedure is restarted at a better solution. In the random improvement approach the procedure is restarted as soon as a better neighbour is found. In (6) Lin recommends the use of the random improvement approach. In our subproblem solvers either approach

can be used but all of the experiments described here were performed using random improvement.

The Lin and Croes subproblem solvers are modifications of the algorithms described in (6) and (7). The modifications are those required to allow the approach to be applied to linear problems. Recall that any linear solver can be used to solve an ordinary circular TSP by identifying the first and last cities in the linear tour.

The Lin and Croes procedures are based on hill-climbing and reduction. The reduction aspect of the procedures is described later in this section.

In the Lin subproblem solver, neighbours of linear tours are other linear tours which can be obtained by removing any three links from a tour and then rejoining the four "pieces" of tours to form some other linear tour. There are 8 possible ways of rejoining a tour from which three links have been removed so that the resulting structure is a linear tour. One of these produces the original linear tour so that only seven possibilities need be considered. Suppose three links in a linear tour join cities i to i+1, j to j+1 and k to k+1. The seven possible ways of constructing a new linear tour when these links are removed are illustrated in figure 5.

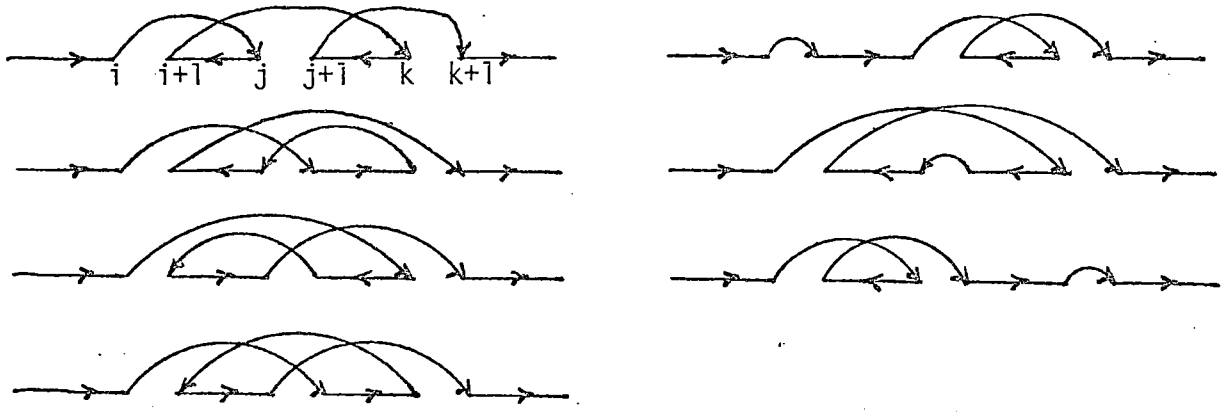


Figure 5: Breaking and Rejoining Subproblem Links.

Any linear tour which cannot be improved by removing k links and then rejoining the pieces to form a new linear tour is called a k-opt tour. The Lin procedure operates by finding a succession of 3-opt tours. The Croes procedure, a weaker but faster subproblem solver, operates by finding 2-opt tours. Because a 2-opt approach does not allow for as many combinatorial possibilities as a 3-opt approach, we would expect 3-opt tours to be better than 2-opt tours. (3-opt tours will always be at least as good as 2-opt tours since any k+1-opt tour is also a k-opt tour.) Experience indicates this to be correct.

The relative importance of 2 and 3-opt tours is illustrated in the following definitions and theorems.

Definition. A tour is inversion optimal if no connected section of the tour can be removed and reinserted at the same place in the reverse order to produce a better tour.

Definition. A tour is insertion optimal if no connected section of the tour can be removed, the break closed, and the section inserted at some other point to produce a better tour.

Theorem A 2-opt tour is inversion optimal.

Theorem A 3-opt tour is both inversion and insertion optimal.

In practice, successive 3-opt tours often have the same links. If the same sequence of 3 cities appears in order in a number of 3-opt tours, then a good heuristic is to reduce the size of the problem by removing the middle city of the sequence and then adjusting the distance matrix so that the other two cities will always be neighbours in any solution to the reduced problem. This reduction process, recommended by Lin in (6) is implemented in both our Lin and Croes subproblem solvers.

The Lin and Croes procedures each go through a number of reduction cycles. In a single reduction cycle a number of 3-opt (or 2-opt) tours

are produced. The problem is then reduced by removing cities with the reduction process. For example, if a sequence of 3 cities abc (or cba) appears in all of the 3-opt tours discovered in a reduction cycle, then city b is removed from the best tour, the distance matrix altered so that the distance from a to b is effectively $-\infty$, and a new reduction cycle is started on a shuffled permutation of the reduced best tour. After a prescribed number of reduction cycles, a complete solution is reconstructed from the final reduced tour. In order to reconstruct a total solution from a reduced solution, a record must be kept of the cities which were removed, and which cities they "followed" in the tour before reduction. Whenever a city is removed from a tour, its name is stored on the "follows queue" for the city in front of it. (The procedure for a linear tour is such that the first and last cities are never removed). The reconstruction process involves reinserting follows queues at the appropriate places in reduced tours. Since a city in a follows queue may itself have a follows queue, the final solution must be reconstructed in several stages.

Both the Lin and Croes solvers can be run with any number of reduction cycles and with each reduction cycle requiring the construction of any number of 3-opt tours. Random initial tours are constructed by shuffling other tours. The shuffling process is carried out through the use of a linear congruential random number generator.

In addition to reduction, Lin also describes a process for producing "almost" 3-opt tours. The process was programmed into our subproblem solvers but was not used in the experiments described later in the report. Its significance is related to certain parts of Lin's circular tour algorithm which rely on the rotation of permutations representing circular tours. Linear tours or solutions have fixed endpoints so that permutation rotations do not result in identical tour solutions, as they do for circular tours. The validity of the "almost" 3-opt process is related to this ability to

rotate circular tour representations and is not really useful when solving linear problems. In general we found it necessary to fully understand and then re-interpret Lin's approach in order to apply it to linear problems. The modifications that were required were more fundamental than those for the modification of Karps algorithm.

The storage requirements for the Lin (or Croes) approach are relatively modest. The role of the Lin procedure as a subproblem solver in our system, however, required extra storage beyond that required when it is used as a "stand-alone" solver. During reduction the intercity distance matrix is altered by the Lin (Croes) procedures. Since the distance matrix is used by other parts of the system, or by applications of the Lin procedure to other subproblems, it is necessary to prevent its permanent destruction by some routine during the interactive solution process. The easiest and computationally most efficient solution to this problem is to have a second, temporary distance matrix in the system. The first thing the Lin and Croes subproblem solvers do is to load this matrix with the distances for the cities in the subproblem to be solved.

The computation time required for the Lin subsolver varies with the problem under consideration. In general, all that can be said is that a 3-opt "check out" requires on the order of $\binom{n}{3}$ computations. A check out takes place when every possible neighbour of a solution is constructed and no better solution is discovered.

For our implementation, the 6 city problems mentioned earlier in connection with the Karp procedure required an average .7 CPU seconds for a Lin solution and .5 seconds for a Croes solution. The 10 city problems required 1.3 and .9 seconds. All three subproblem solvers produced the same solutions for these problems. The differences between the algorithms become really significant for larger problems and can be observed in the figures given for the examples in the following sections. In all of the sample

problems described in this report the Lin and Croes procedures were run at their default settings of 2 reduction cycles and 2 k-opts per reduction cycle.

7. System Commands and Algorithms

In addition to the subproblem solvers, several other interesting algorithms are used in the interactive system. In this section we will first describe the basic solution commands. The algorithms which are used to carry out several of these commands will then be described in some detail.

(a) Display Commands. Display commands come in pairs of erase and display commands. These commands can be given either directly by the user or indirectly by issuing a command which calls one or more of these commands.

(i) DCITIES, ECITIES will cause the points representing the cities on the plane to be displayed or erased.

(ii) DSUB(X), DSUBS, ESUBS will cause polygon boundaries defining subproblems to be erased and displayed. Any display command requiring an argument can be either given the name of an object or a *. If the argument is a * the routine assumes the user will identify the object argument by pointing at the object on the display. When an argument is a *, the routine for reading the RAND pen and searching through the data structure is activated.

(iii) DSOL(X), DSOLS, ESOLS, will cause subproblem solutions to be displayed and erased.

(iv) DSYNTH(X), DSYNTHS, ESYNTHS will cause synthesized solutions to be displayed and erased.

(v) DSUBNAMES, DCITYNAMES, ESUBNAMES, ECITYNAMES, will cause displayed subproblems and cities to be labelled with their object names. The erase routines will erase these labels.

(vi) EGARBAGE will cause miscellaneous other displayed information to all be erased.

(b) Solution Process Commands. These commands automatically call certain display commands. These commands generally come in pairs of create and kill commands

(i) CSUB(X), KSUB(X) will allow the creation of or will kill a subproblem. For the CSUB command if X is a * the system will automatically generate a name for the created subproblem. Otherwise the user suggested name is used. For the KSUB command the same conventions are used as for the display commands.

When the CSUB command is initiated the user is expected to draw a polygon on the display screen using the RAND pen. The computer then determines which terminals and/or non-terminals in the present context have centroids inside this polygon. These objects are removed from the context non-terminal's extension and become the extension objects of the new subproblem non-terminal. The new non-terminal is then added to the (diminished) extension of the context non-terminal.

When a subproblem or non-terminal is killed, it is removed from the subproblem structure. In particular, it is removed from the extension of the non-terminal to which it belongs. The objects in the extension of the killed non-terminal are then added to the extension of the non-terminal to which the killed non-terminal belonged. During both creation and killing,any affected solution structures (e.g. subproblem solutions or synthesized solutions) are automatically deleted since they are no longer valid. Recall that these solution structures exist as parts of extensions of non-terminals.

(ii) CKSOL(X),CLSOL(X),CLSOL(X),KSOL(X). The first three of these commands will create subproblem solutions to the named subproblems using the Karp, Lin or Croes subproblem solvers. KSOL will delete or kill the solution to a named subproblem. These commands provide real time feedback to the user by calling appropriate display commands. All of the subproblem solvers will

create circular tours for the top level subproblem UNIV and linear solutions to all other subproblems. When solving linear problems they will request endpoints or subproblem boundary points to be specified. The user can then point to the two subproblem objects he wishes to be the path endpoints in the subproblem solution.

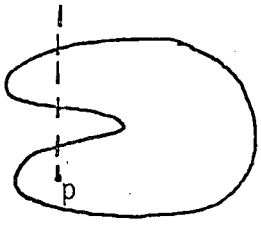
(iii) CSYNTH(X), KSYNTH(X) will create and kill a solutions synthesis. CSYNTH(X) will snythesize the solutions to the subproblem X, the subproblems of X (i.e. the structure of the non-terminal objects in the extension of X), the subproblems of some subproblem of X, and so on. KSYNTH will delete a solution synthesis (i.e. delete the global structure) from a non-terminal.

(c) CSUB Algorithms. The operation of CSUB requires algorithms to read points from the RAND tablet pen, create a subproblem polygon from these points, and to then search through the non-terminal tree to determine which centroids of the objects in the extension of the present context lie inside the polygon. With the exception of the algorithm for determining when a centroid lies inside a polygon these algorithms are technical and uninteresting in nature. The exception is described below.

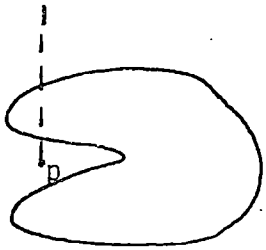
One technique used in determining if a point p=(x,y) lies inside or outside a simple closed curve C (eg. a polygon) is to draw a half infinite line h from p to (x,∞) and count the number of times c that h crosses C. If p is inside C we might expect that c will always be odd and that if c is even p will be outside C. The intuitive argument for this reasoning is illustrated in figure 6.

Unfortunately this simple decision rule, discovered independently by the author and others,fails in several situations. The difficulties arise when the line h does not cross some portion of the curve but is tangent to it. For the case where C is a sequence of straight line segments this occurs when p lies beneath a corner (vertex) or a vertical line segment. In the following

discussion we will assume that C is a collection of line segments although all of the arguments can be extended for continuous (in the derivative) curves C.



c=3 p inside C



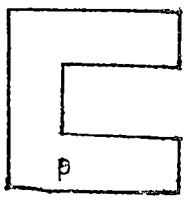
c=2 p outside C

Figure 6.

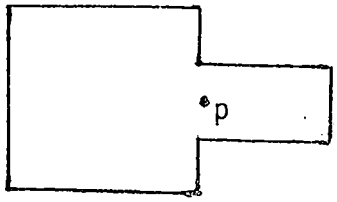
The following examples illustrate that no simple variation of the decision rule will correct its deficiencies.

Example 1.

Suppose we decide to count 1 whenever h crosses a vertical line segment and that the line segments in C (i.e. the sections of C between adjacent corner points) are ordered in clockwise fashion. Each corner point "belongs" to the following line segment. The values of c for the points in figure 7 are 3 and 2 (odd and even) and yet in both cases p is inside C. Clearly counting zero instead of 1 will not change the situation.



c=3

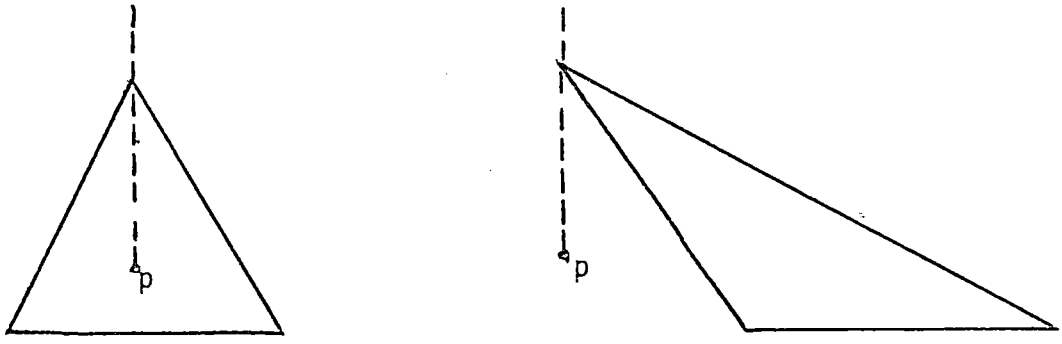


c=2

Figure 7.

Example 2.

A similar situation arises in the treatment of corner points alone. Suppose, as in example 1, that a corner point is considered to belong to the following line. The values for c in the points in figure 8 are both 1 and in one case p is inside C and in the other case outside C.



c = 1 c = 1

Figure 8.

It is easy to see that no simple counting variation of the ways of treating corner points and vertical line segments illustrated in the examples will resolve the problems. Similar examples can be produced when corners are considered to belong to both segments or of different counts c are used in the exceptional situations. We introduce an approach here which can be used to solve these problems and construct a general purpose algorithm.

The approach is based on the following observation. Suppose a point p lies below a corner. Then the correct count for p with respect to that corner is the same as for a point p' which lies one unit to the right of p. This observation is easy to prove for the discrete case where the curves are in fact finite sequences of points on a grid. For the theoretical or continuous case it requires that C be locally connected. It is possible to convince oneself of the validity of this observation by considering the examples in figure 9.

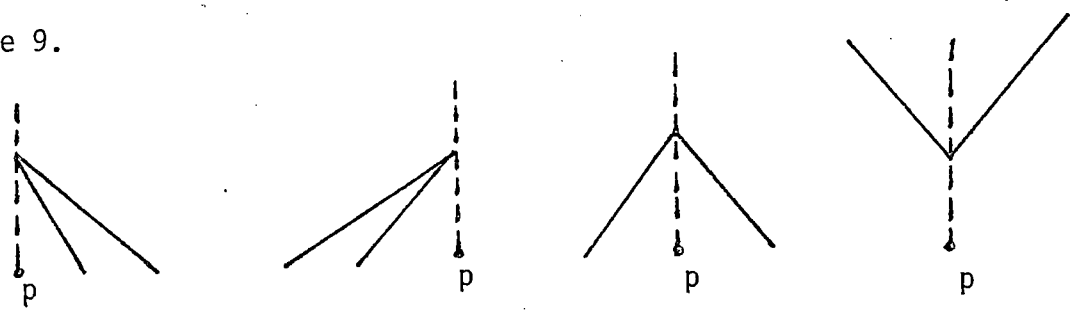


Figure 9.

In practice the algorithm must be able to deal with the case where one of the line segments forming a corner is vertical. The operation of the algorithm is such that this can only be the second line segment. When this occurs the vertical line segment is replaced by a line segment extending from the corner to the second endpoint of the next non-vertical in line segment. The reasoning behind this process can be understood on the basis of the illustrations in figure 10.

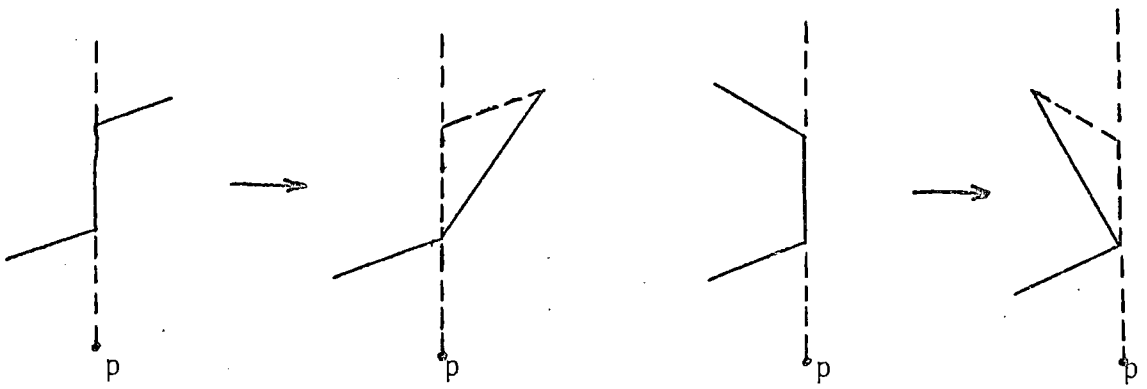


Figure 10.

The algorithm operates as follows:

(i) Each line segment h of C is considered in turn. If h is a vertical line segment the next segment is chosen.

(ii) If $p = (x,y)$, the point in question, is on h the decision has been resolved.

(iii) Suppose the endpoints of h are (x_1,y_1) and (x_2,y_2). If $x_1 < x < x_2$ then the two-point form for a line is used to calculate the y-coordinate z of the point (x,z) on h. If $y > z$ a count c is incremented by 1. The algorithm continues with the next segment on C unless all segments have been considered.

(iv) If ($x \leq x_1$ and $x < x_2$) or ($x \geq x_1$ and $x > x_2$) then the next line segment, unless all segments have been considered, is selected.

(v) If $x = x_2$ (i.e. x is directly below the second vertex defining h) then:

(1) The next <u>non-vertical</u> line segment h' in C is chosen. If there are no remaining non-vertical line segments the algorithm reinitializes the list of remaining line segments to C for this one step. After a non-vertical line segment is found C is considered to have been exhausted for all other steps in the algorithm.

(2) One unit is added to the x-coordinate of p and the line segment h" which extends from (x_2,y_2) to the second endpoint (x_2', y_2') of h' is formed. The count is now modified for the relationship between the modified point $p' = (x',y') = (x+1,y)$ and the two line segments h and h". If $x_1 \leq x' \leq x_2$ or $x_2 \leq x' \leq x_1$ than the point (x',z) on h is calculated. If $y' \leq z$ then one is added to the count. If $x_2 \leq x' \leq x_2'$ or $x_2' \leq x' \leq x_2$ then the point (x',z')on n" is calculated. If $y' \leq z'$ then one is added to the count.

(vi) If all of the segments in C have been considered after the completion of any step then the algorithm returns a positive or negative result depending on whether the current count is odd or even.

The proof that this algorithm is correct depends on the discrete case analogy of the Jordan Curve Theorem and the following theorem. The following theorem is the basis for the "point adjustment" process.

Theorem Suppose the "line" h is a finite sequence of discrete points on a grid G and that p is a point on a grid G' at least as fine as G. Suppose that h does not have slope zero. Then if

(i) p=(x,y) is to the left of h, p'=(x+1,y) is either to the left or on h.

(ii) p=(x,y) is to the right of h, p'=(x+1,y) is either to the right or on h.

The theorem guarantees that a point will not "hop over" a line when it is moved one point to the right. The continuous form of the theorem is:

Theorem Suppose p=(x,y) lies to the left of a curve segment C. Then there exists $\epsilon > 0$ such that if

(i) p(x,y) is to the left of C, so is p'=(x+ϵ,y)

(ii) p (x,y) is to the right of C, so is p'=(x+ϵ ,y).

(d) CSUB and extension intersection. We note here an important side effect of the CSUB algorithm that results when objects are removed from the context non-terminals extension and added to the extension of the new non-terminal. Because these objects are removed from the extension at the time that the new non-terminal is created the intersection problem is avoided. Two non-terminals intersect if they have extension objects in common. In our system two non-terminals can never intersect. This avoids a tangle of difficult problems which would otherwise arise during solution synthesis. In some problem solving situations it may be necessary to allow intersection (cf. the discussion of subproblem independence in (5)). In

the related algorithms for cluster detection the use of overlapping "non-terminals" appears to help avoid the "migration problem" (6). Our present experience with the TSP system indicates that any desirable facility which might require intersections can be more easily implemented in some alternative way.

(e) CSYNTH algorithm. It might appear, at first glance, that all CSYNTH needed to do was to simply join together the lowest level subproblem solutions in the order indicated by the subproblem solutions in the higher levels of the subproblem tree. This is not so; consider the example in figure 11.

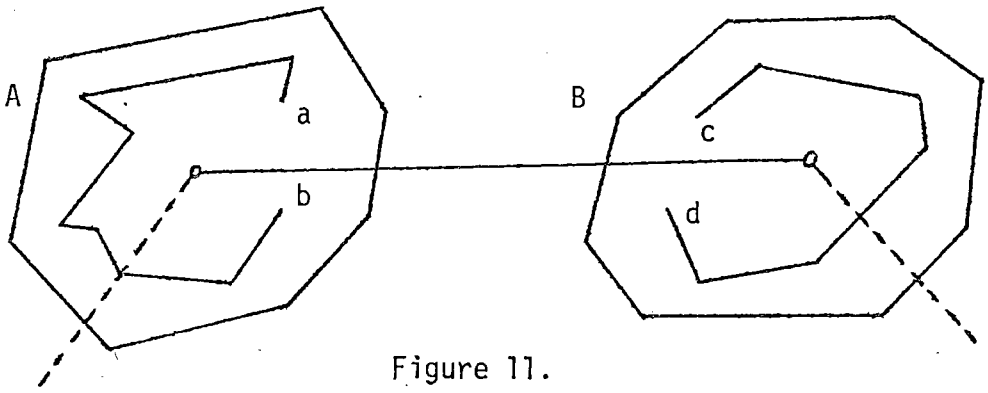


Figure 11.

The small circles are the centroids of non-terminals A and B and the dotted lines indicate the non-terminal structure (i.e. subproblem solution order) relating A and B as extensional objects in the next higher level subproblem. When A was solved it is likely that there was no information to determine whether the solution ran from a to b or from b to a. Hence, depending on whether a is to be joined to c or to d or whether b is to be joined to c or to d, it may be necessary to reverse the direction of the linear subtour between a and b. CSYNTH presently uses a simple heuristic to determine how to choose which endpoints in adjacent subsolutions should be matched when the subsolutions are joined together. On the basis of this heuristic it can determine if a subsolution tour must be reversed before it is joined to other subsolution tours.

The CSYNTH algorithm operates by first going through the subproblem tree to be synthesized and reversing the direction of any subproblem list of objects that is in the wrong order. For example, if the solution to A in figure 11 runs from a to b yet the heuristic indicates that the synthesis should be of the form

b,...,a,c,...,d

then the subsolution is reversed. When all subsolutions at all levels are in the correct order they are synthesized.

The synthesis heuristic for determining which pair of endpoints in adjacent solutions should be matched - and hence for determining whether a subsolution must be reversed - has consistantly agreed with the users intuition. It is defined as follows. Suppose B is the non-terminal whose solution direction must be checked.

(i) Let A be the first non-terminal before B and C the first non-terminal after B in the subsolution to the subproblem to which B belongs. Let c_A and c_B be the centroids of A and B and c_x and c_y the centroids in the first and last objects in the subsolution to B. Let DIS be the intercity distance function. If

$$DIS(c_A,c_y)+DIS(c_x,c_C) < DIS(c_A,c_x)+DIS(c_y,c_C)$$

then reverse the direction of the subproblem list for B.

(ii) If there is no non-terminal before (after) B in the subsolution to which B belongs (recall that subproblem solutions are linear) then let A (C) be the first non-terminal before (after) the non-terminal to which B belongs in the subproblem solution which includes B. If A or C is still not defined the process is tried one step higher up in the subproblem tree. (See example below.)

(iii) If there is no non-terminal before B at any level in the subproblem tree being synthesized, then the following rule for re-ordering is used. If

DIS (c_x, c_C) < DIS (c_y, c_C) then the direction is reversed. A similar rule can be used if there is no non-terminal after B. If we assume that subproblems always have at least two objects, there will always be either a non-terminal before or after every non-terminal B.

Example Suppose A B C is the subproblem solution to the subproblem at the top of the tree for which synthesis has been requested.

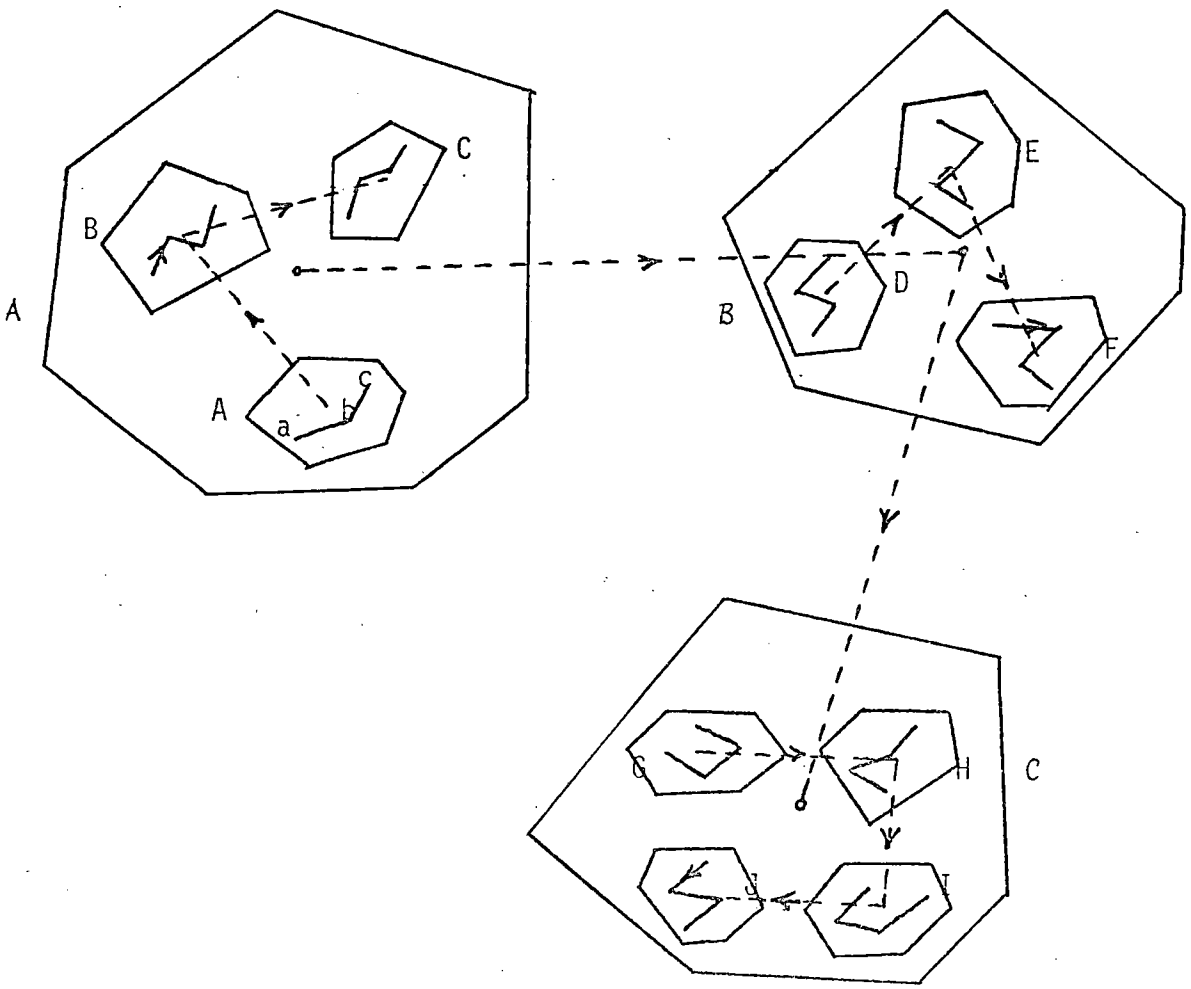


Figure 12.

The solutions to subproblems AB and C all run in the correct direction. The solution to subproblem F will be found to be ordered in the correct direction. c_E is the centroid before F and c_C the centroid after F. J's solution is pointing in the wrong direction. There is no non-terminal after J and c_I is the centroid before I. B's solution is pointing in the correct direction. c_A is before B and c_C after B.

Once the correct directions have been established the synthesis can be easily created by joining together adjacent subproblem solutions in the order indicated by the next higher level subproblem solution. This is equivalent to writing down the top level subproblem solution and then rewriting each non-terminal by its solution list and continuing this process until no non-terminals remain.

Example

ABC → ABCBC → ABCDEFC

→ ABCDEFGHIJ → abcBCDEFGHIJ→....

8. Sample Problems and Problem Input

In order to test the operation of the system and hence experiment with our subproblem or planning approach it was necessary to devise a set of sample problems. We needed problems which had patterns which suggest subproblem possibilities to a human. In order to avoid rigging the experiments input was required which exhibited different patterns and yet was still randomly chosen in some way.

The first idea was to construct a random point generator which could be "guided" by a pattern randomly chosen from a repetoire of patterns. One method would be to have the plane divided into a hierarchy of subsections. The subsections could then be referenced by a tree of the form in figure 13.

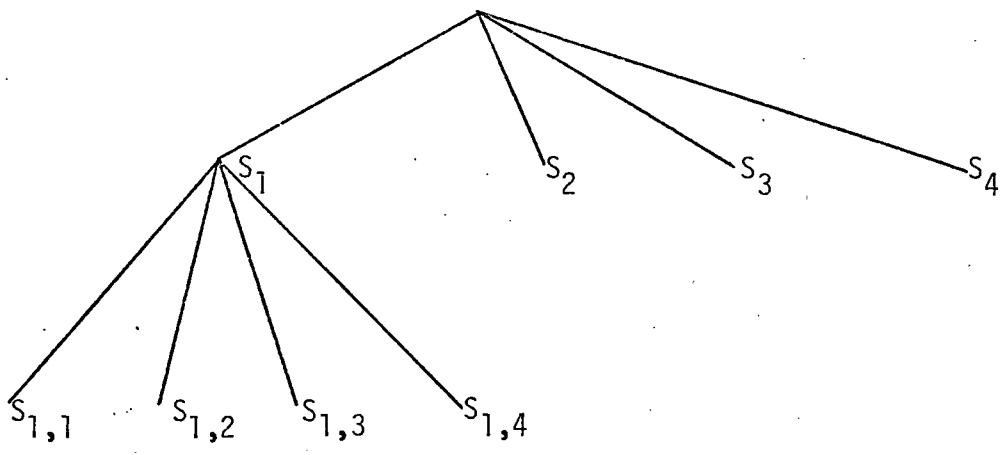


Figure 13.

In this "quarternary" approach, the node S_1 refers to a quarter section of the plane, $S_{1,2}$ refers to a quarter section of S_1, and so on. At the terminal nodes are references to small subsections of the plane. Suppose a probability is attached to each node in the tree. Then a point can be generated by traversing the tree according to the probabilities at the nodes and then generating a point randomly in the arrived at terminal subsection. Such a Markov tree can be used to guide the random generation of points with some pattern. The guiding pattern is determined by the probabilities "loaded" into the tree.

The difficulty with this approach is in deciding which idealized "Markov patterns" should be used. We did not know when we began these experiments

what such patterns would be. In fact one goal of the research effort was to determine if, at least for this problem (TSP), such "high-level" solution subproblem ideas existed. To overcome this difficulty we adopted a very simple approach. The sample problems are chosen from a political atlas of the world (8). To input a problem a page of the atlas is placed on the RAND Tablet and the city coordinates input using the RAND pen. In this way we were able to conduct experiments with random data exhibiting interesting patterns.

In the following three sections, detailed accounts are provided of our experience using the system for three different problems. The only criteria in choosing the first two problems was that they exhibit some kind of structure. We did not know when we chose these problems just what the structure was or how it would be used to solve the problem. The experiments were to determine if the user: (i) could easily express and try out any ideas for a solution he might have and (ii) would in fact have good ideas. The third problem was chosen because it did not appear at first to have any patterns or structure to it. We wanted to experiment with such a problem as well to determine, at least for the TSP, the limitations of this approach.

9. The South American Travelling Salesman Problem

Two "high level" or abstract strategic ideas occurred to us for the solution of this problem (figure 14). The first was to construct a simple circular tour of subproblems. We imagined a tour following the coastline, so to speak, which made excursions inland whenever necessary to pickup "stray cities". The effectiveness of the subproblem approach in this case was to sketch out such a solution, forcing stray cities to be associated with the most appropriate subset of coastline cities. We will distinguish the exploration of this idea as Phase I. Other phases will be concerned with the exploration of other ideas.

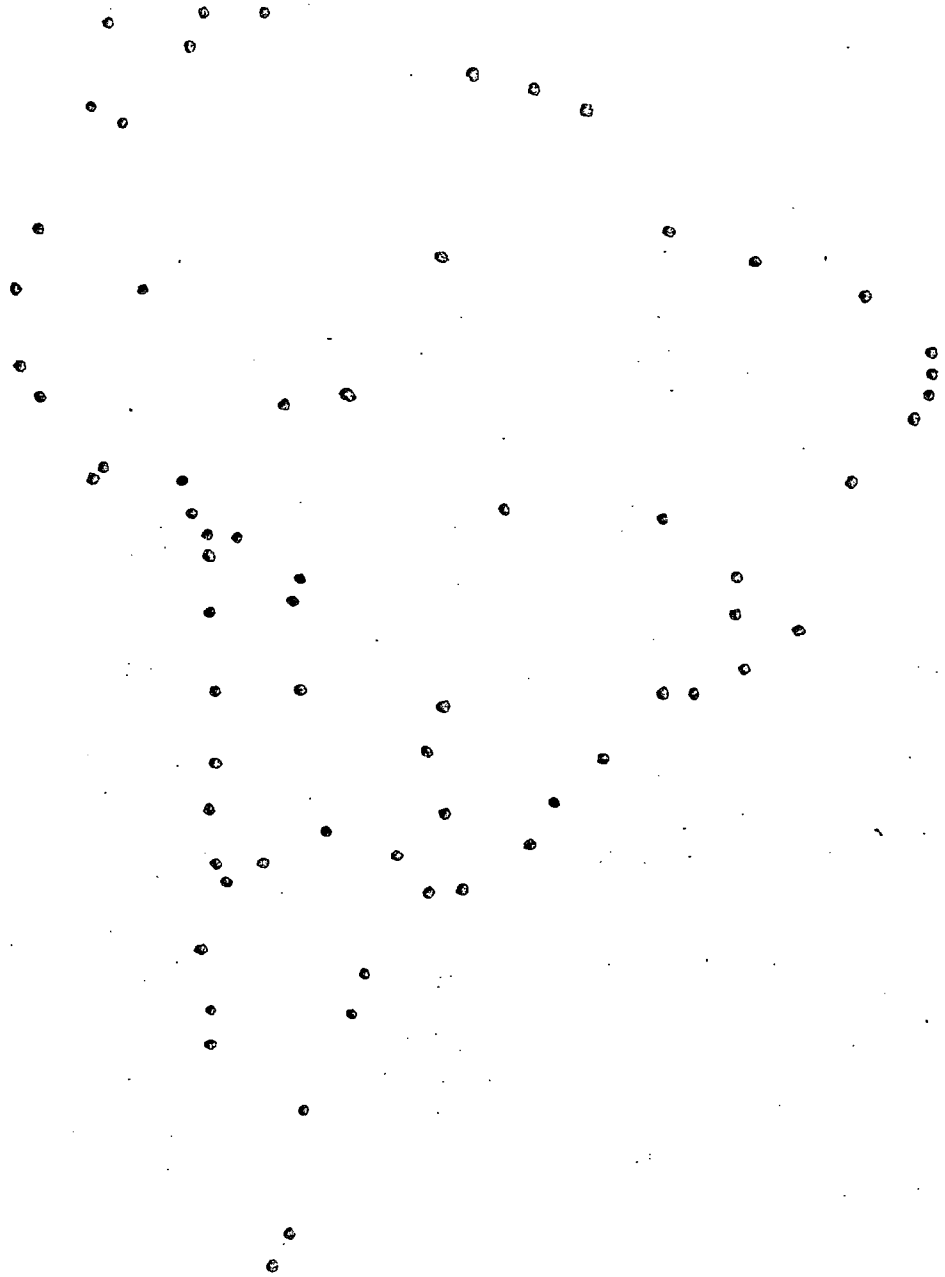


Figure 14: The South American Travelling Salesman Problem

Phase I In figure 15 we have created three obvious subproblems and called on subproblem solvers for their solutions. This required three CSUB commands and three solution commands.

The first two subproblems created were the "tip" and upper right "shoulder" groups of cities. These groups were created and solved because we knew what their solutions should be and how they should fit in with the rest of the problem. Since we knew what the solutions should be we used the fast heuristic Lin supbroblem solver (CLSOL). If this solver had not returned the expected solution we might check both CLSOL and our intuition by calling on the exact Karp subsolver (CKSOL).

A deficiency in the system recognizable even at this early point is the inability of the user to easily suggest his own subproblem solutions (e.g. with a "manual" subproblem solver that might be called CUSOL - create user solution). For all low level subproblems (terminal objects), however, it is easier and faster for the user to call on a fast heuristic procedure which can be relied on for small easy problems than for the user to input all the pieces of a subsolution. The addition of CUSOL to the system would not be difficult; it is a natural feature of the planning-subproblem structure approach.

Since there were two "obvious" routes to choose from for the third subproblem (the one in the middle in figure 15 between the other two) it was solved using CKSOL. The choice of boundary points for the third subproblem (i.e. the endpoint cities that will be joined to cities in other subproblems) was not entirely obvious. We decided to put off making up our mind about the boundary points for this subproblem and killed the subproblem solution we had just created. One of the problems was that we were not yet decided on how to treat the two stray cities to the left of this subproblem.

We now created three new subproblems (figure 16). In creating two of

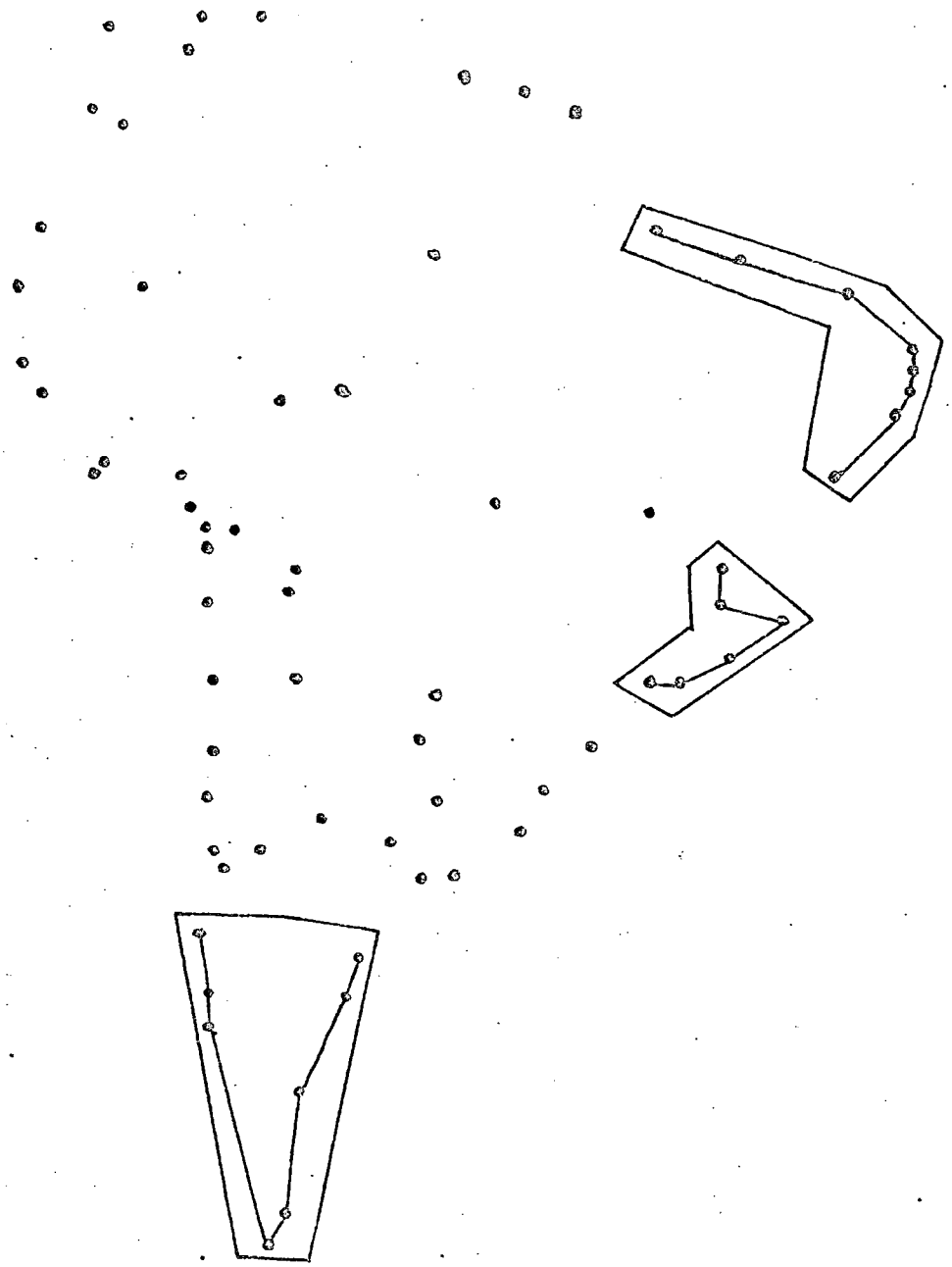


Figure 15: Creating and Solving Some Obvious Subproblems

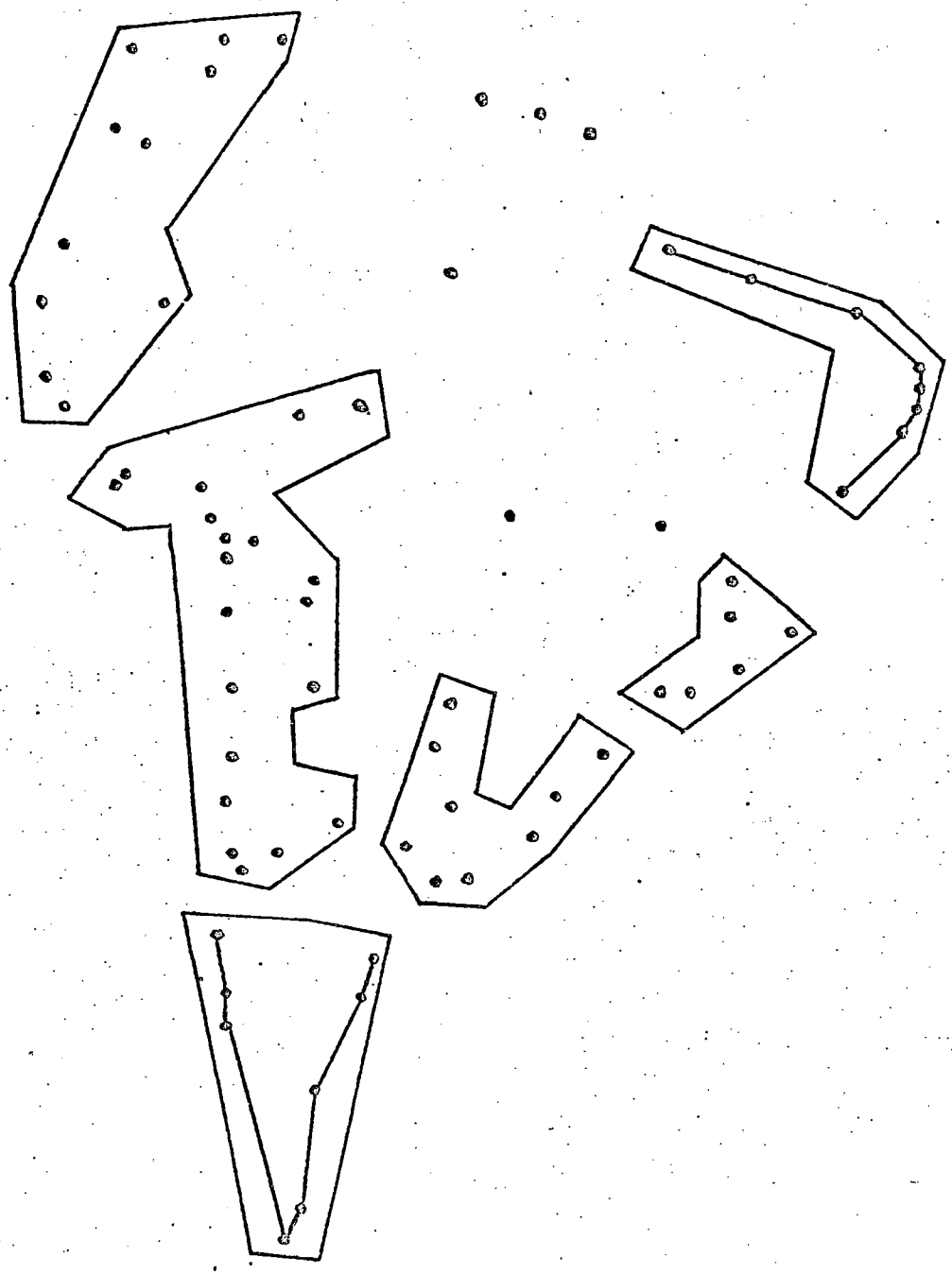


Figure 16: Deleting a Subproblem Solution and Creating More Subproblems

the subproblems (SP5 and SP6 in figure 19) it was not obvious which subproblem one of the cities should be in (C40 in figure 17). The city in question lies on the "border" between these two subproblems. It was decided to include it in the left-most subproblem, SP6. An extended facility would be a feature which allowed the user to check-point the partially developed solution when such alternatives had to be decided upon so that he could easily return and pick up an unexplored alternative.

We now observed that we could almost create a Karp solution to the top level subproblem. "Almost" because the number of objects at the top level exceeded by 1 the maximum allowable subproblem size for CKSOL. We therefore created and solved the obvious subproblem of the three cities in a row at the top of the set of cities and then called CKSOL for the top level subproblem (figure 18). The top level subproblem was solved in order to provide a picture of the way in which the subproblems would be joined together. On the basis of this information we would be able to choose boundary points for our solutions to the remaining subproblems.

To assist in the manipulation of the subproblems we chose to display the internally generated subproblem names (figure 19). In all of the experiments described in the report the system was allowed to choose all the subproblem names.

At this point the top level solution did not agree with our intuition. We suspected that the stray city below SP7 should be visited on the way from SP7 to SP2 rather than on the way from SP4 to SP7. This was probably the result of the systems having represented large subproblems by centroids in constructing its solution. Our experience was that the user will often want to construct his own solutions for high level subproblems. It is, however, relatively easy to manipulate the subproblem to get what you want. In order to maintain control of the route through the city in question, the top level subsolution was killed, SP7 killed and SP8 created and the

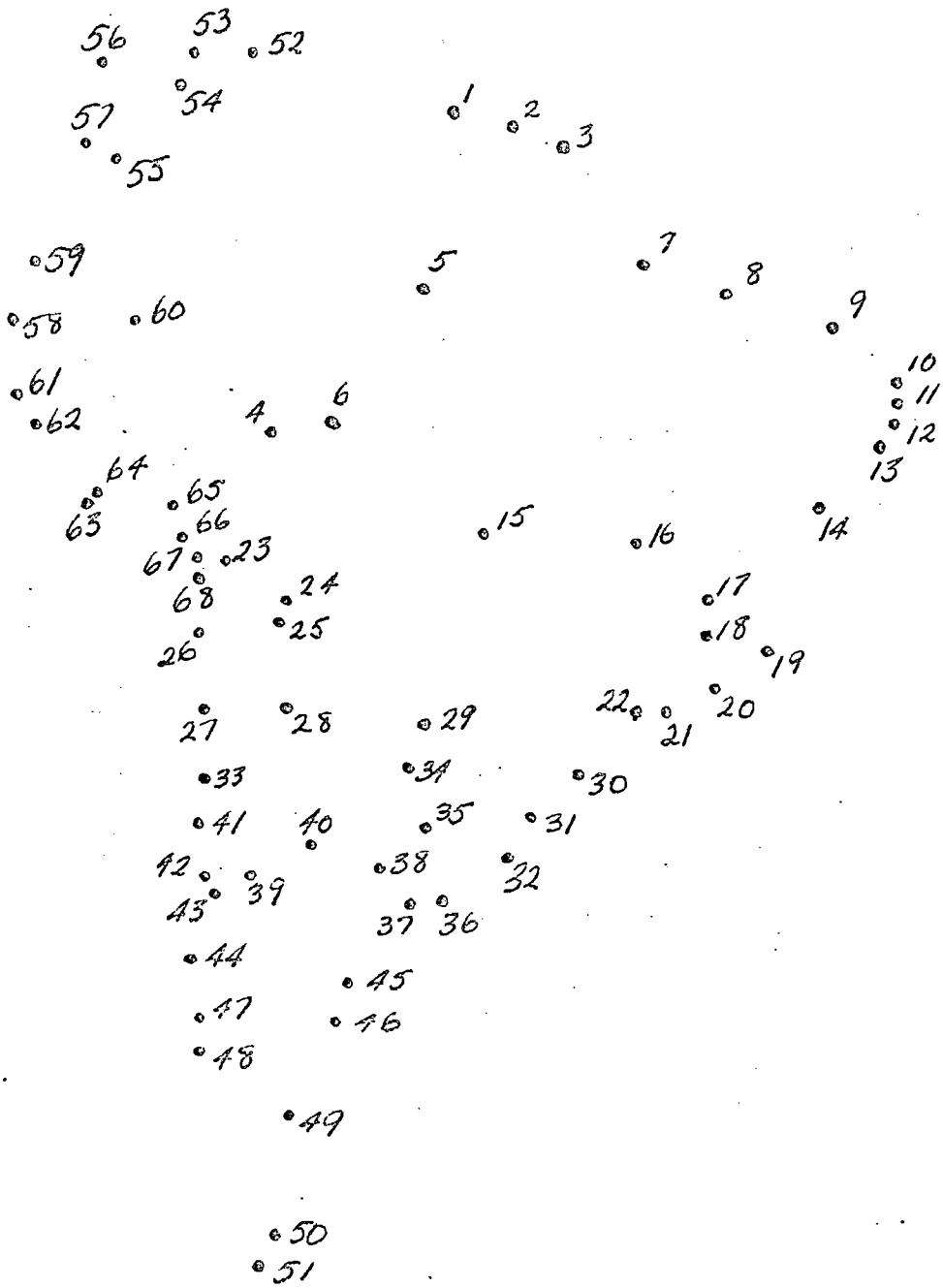


Figure 17: City Numbers

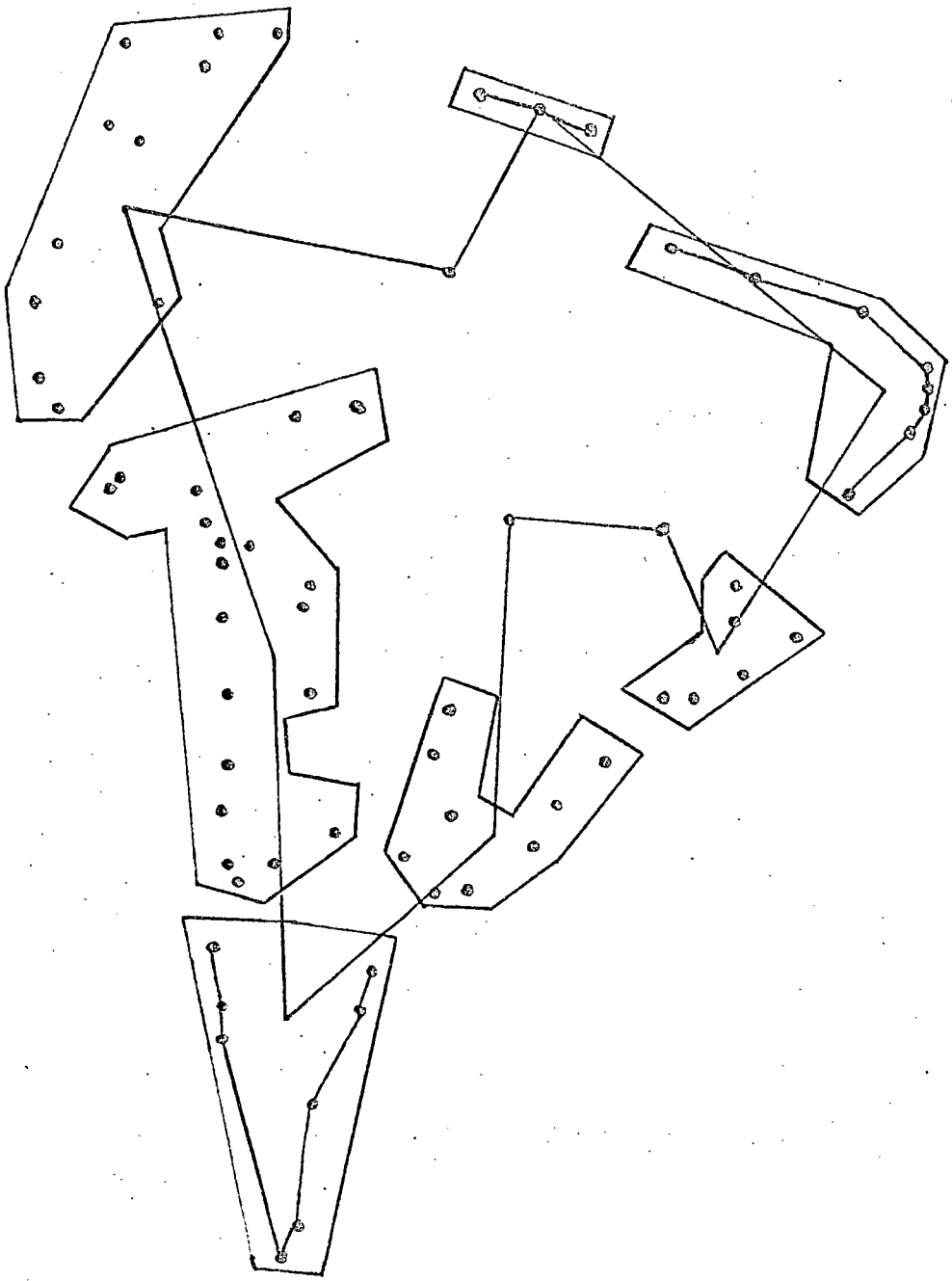


Figure 18: Creating and Solving a Subproblem and Solving the Top Level Subproblem

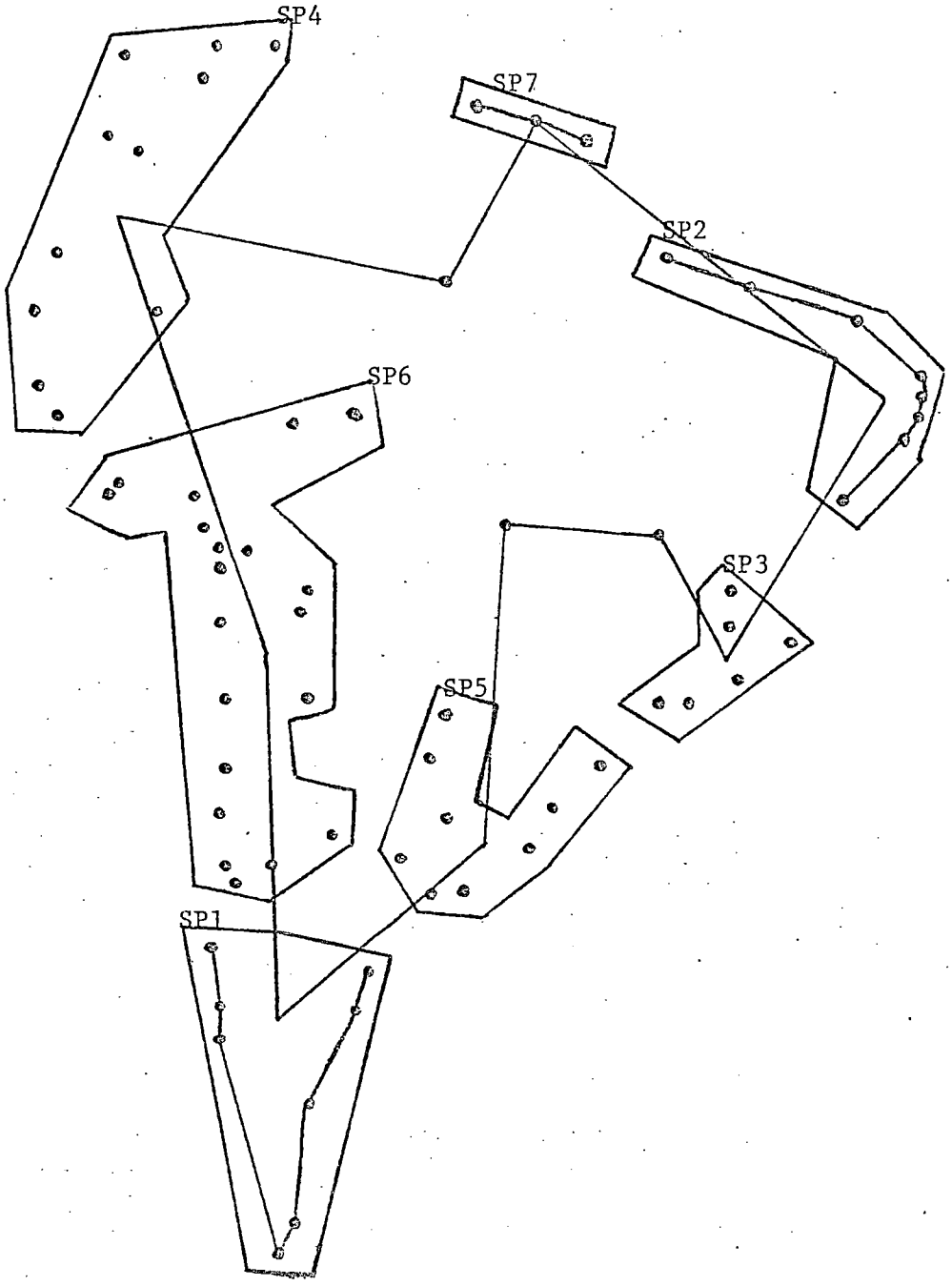


Figure 19: Displaying the Subproblem Names

top level subsolution recomputed (figure 20).

We now decided that visiting the stray cities to the left of SP3 on the way from SP5 to SP3 was questionable. We might either want to visit them on the way from SP3 to SP2 or even during a tour through SP3. We therefore removed these two points from the top level problem by killing SP3 and creating SP9 (figure 21).

At this point solving all the subproblems and then forming a solution synthesis seemed to be a good idea. We first reduced the subproblem size of SP6 by creating the sub-subproblem SP6 (figure 21).

The first subproblem chosen for solution was SP6. Although it is clear which object of SP6 should be the endpoint that interfaces with SP1 it is not clear which city should be the endpoint that interfaces with SP4. It is easy to try different solution possibilities by successively creating and killing solutions for different choices of endpoint pairs and then choosing the best of these (every subproblem solver returns the cost of the solution it constructs). The difficulty is that different choices of endpoints for SP6 are not independent of the choices of endpoints for SP4. We do not want the optimal subsolution of SP6, but the subsolution which is optimal with respect to its own cost plus the cost of "joining" that subsolution to the subsolution of SP4. We call this the context problem.

Each of two alternative facilities in the system would solve the context problem. One would allow the user to ask for the subproblem solution which was optimal with respect to its own length plus the lengths of the links to two stated endpoints in neighbouring subproblems. A second approach would not require that the user specify endpoints for the subproblem in question, but only the associated "linking endpoints" for the two neighbouring subproblems. The subproblem solver would then solve a larger subproblem consisting of the original subproblem plus these two outside endpoints.

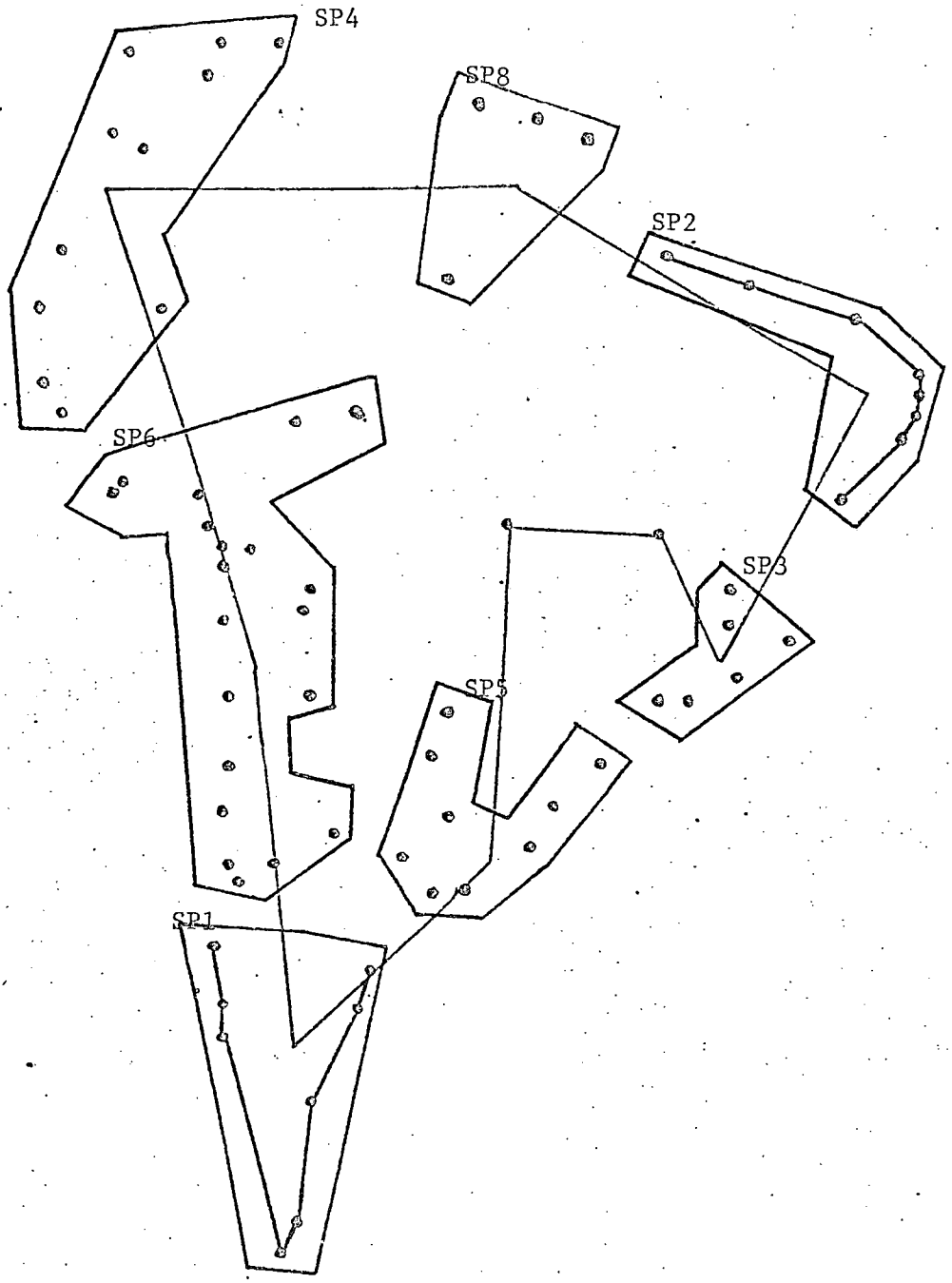


Figure 20: Deleting Top Level Solution. Changing a Subproblem and Re-solving Top Level Subproblem

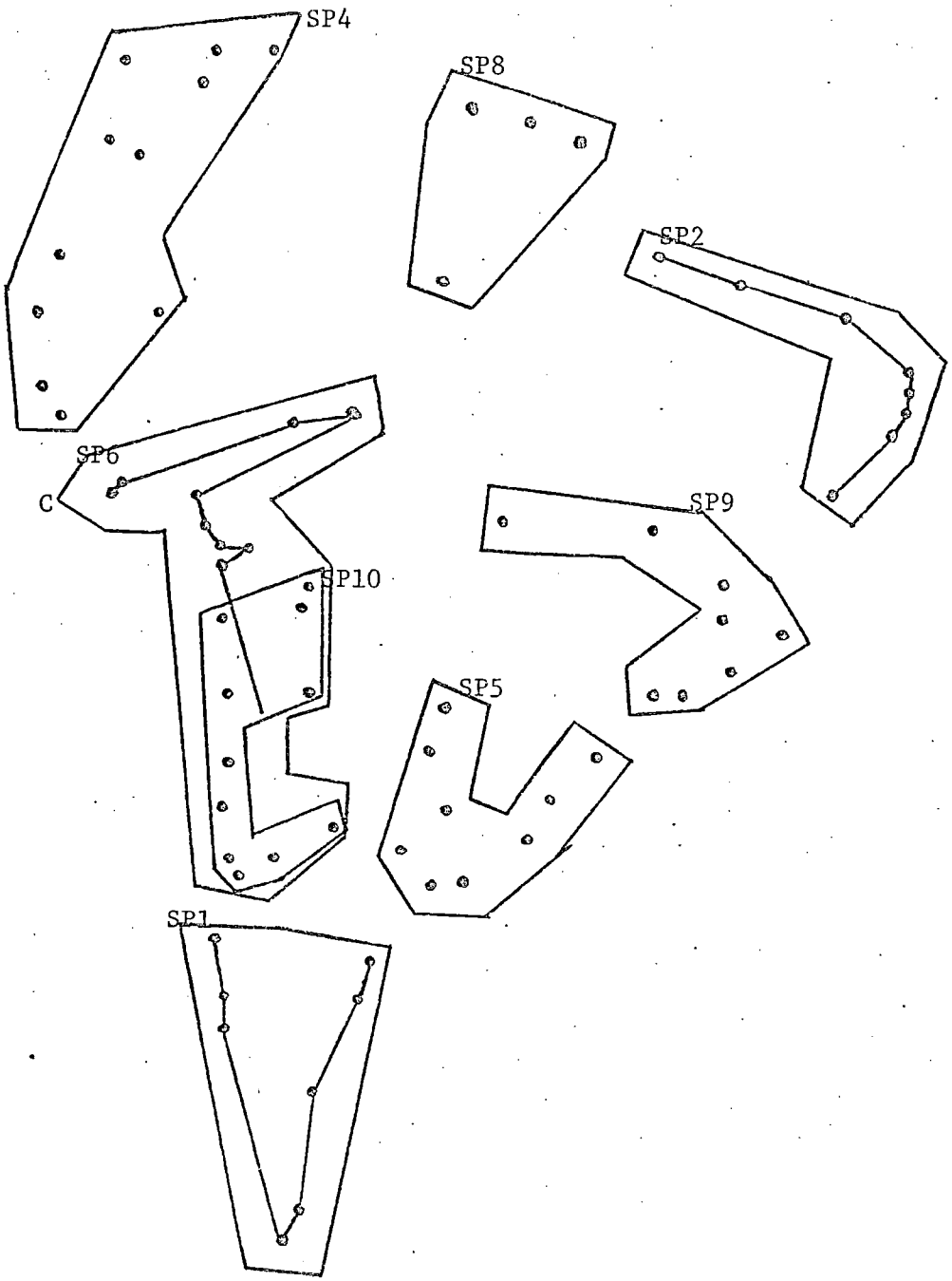


Figure 21: Deleting Top Level Solution. Changing a Subproblem and Creating a Sub-subproblem. Solving a Subproblem. Displaying New Subproblem Names.

The first alternative is probably the cleanest and would only require the programming of several new subproblem solution commands into the system. Each alternative could be implemented in a way that would allow the user to suggest a list of alternatives. The solver would then choose and return the solution to the best alternative. At present the user must try each alternative out on his own.

The first choice of endpoints resulted in the solution to SP6 shown in figure 21. The exact Karp procedure CKSOL was used here. The second choice for a top endpoint to SP6 resulted in the subsolution in figure 22. This solution was more expensive so we returned to the original solution (figure 23).

Although it would have been a convenience to have had the context facilities for SP6 it turned out that they were not necessary. The best choice of a top endpoint for an optimal subsolution to SP6 is also clearly the best endpoint for the link to SP4. All we needed to know was the endpoint for producing the optimal subsolution of SP6.

We now decided to go ahead and solve all the other subproblems, with the understanding that the top level solution will be the circular tour SP4 SP8 SP2 SP9 SP5 SP1 SP6 SP4. SP4, SP5 and SP6 were solved by choosing endpoints that were closest to endpoints in neighbouring subproblems. In each case this was the obvious choice. Since all of these subproblems were somewhat cluttered and the optimal solution not obvious, CKSOL was used.

For SP8 it seemed that going down to pick up the stray point was probably best accomplished on the way from SP8 to SP2. Consequently, the left-most of the three points was chosen as one endpoint and the stray point as the other. CLSOL was used. With a little thought it is easy to convince oneself that this is the best choice of endpoints. It would have been convenient if it had been easily possible to compare the sum (solution length + interproblem link) for the two endpoint choices for the interface

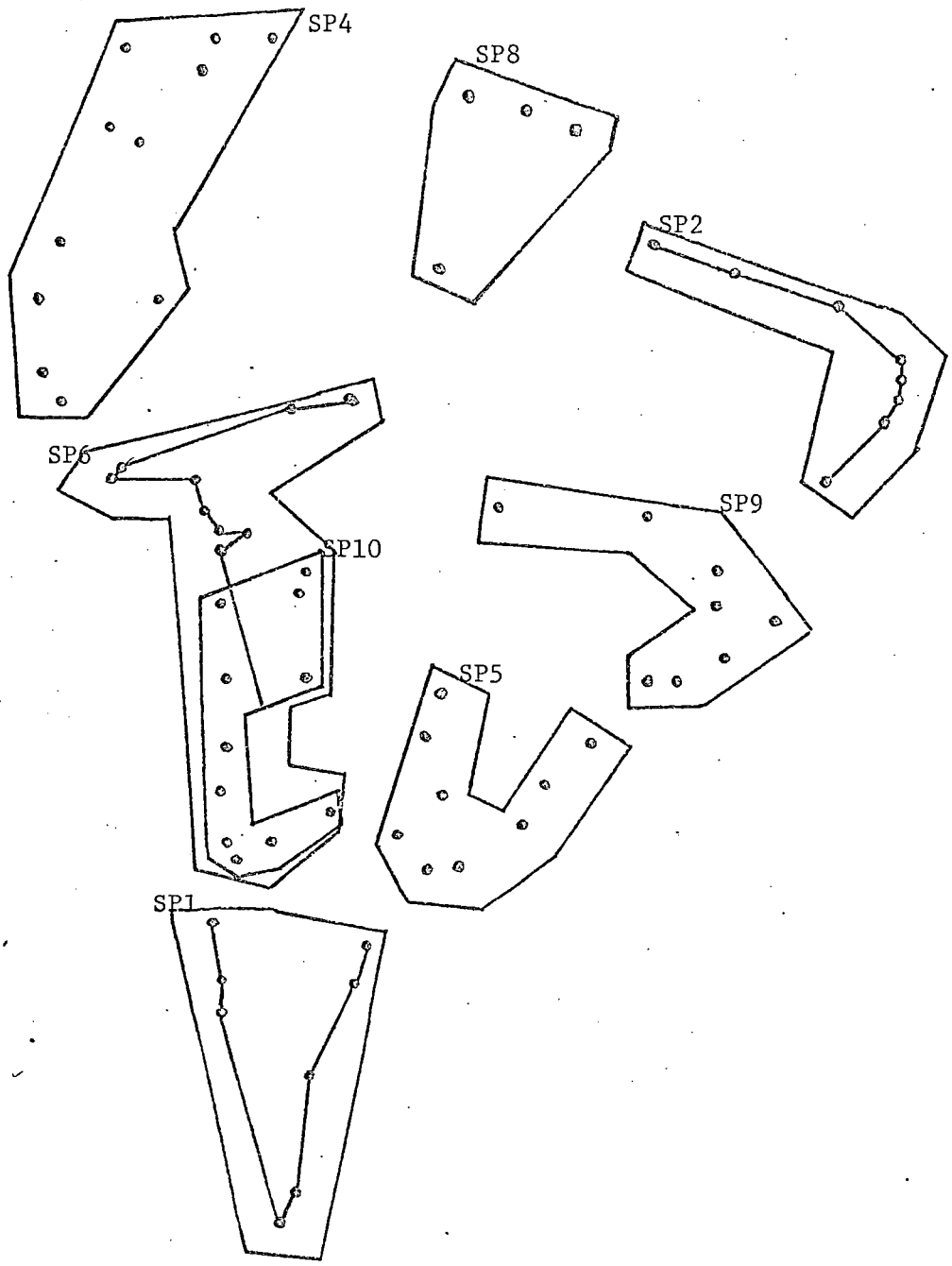


Figure 22: Trying a Different Subproblem Solution

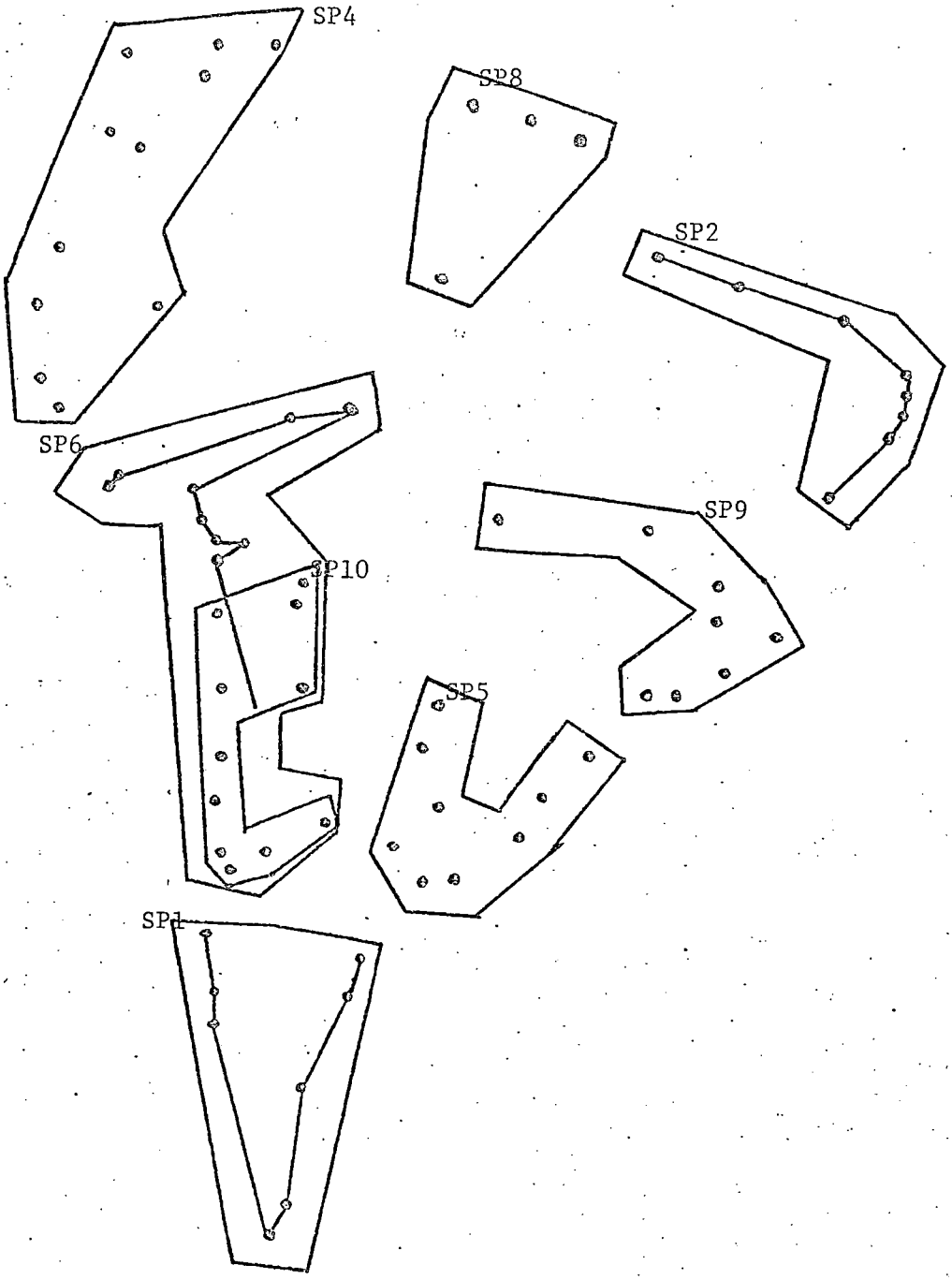


Figure 23: Recreating Original Subproblem Solution

between SP8 and SP2. It is, of course possible to do this "manually" by computing all the factors individually with presently available commands.

The context problem arose again for SP9. We could have compared the different choices manually but decided to just settle on the right-most of the two stray points to the left. CLSOL was used. (figure 24)

Although we had not experimented with several alternative solution decisions we decided to go ahead and look at the final solution we had built. This involved solving the top level subproblem (figure 25) and creating a top level solution synthesis (figure 26). In figure 27 we displayed just the synthesis by erasing subproblem polygons and boundaries. The subproblem solutions were automatically erased (but not killed) by CSYNTH.

CSYNTH returned a value of 2811 for this solution. The "CPU clock" revealed that we had control of the CPU for 174 seconds and we had been sitting and solving for about 40 minutes. This does not include the time required to keep this journal of the problem-solving experience.

Phase II We felt at this point that the solution in figure 27 was representative of the set of possible "coastal tours". The large number of significant excursions in figure 27 indicates that there are perhaps two circular tours in this problem: a coastal tour and an inner tour made up of the so-called stray cities. With the use of two tours, all of the excursion links can be replaced by a single pair of links joining the two tours. If the savings in excursion links outweighs the extra cost of the links to form an inner tour then the "two tour" approach will be better.

In order to reconstruct the subproblem structure to carry out the new idea it was first necessary to kill some of the subproblems created during

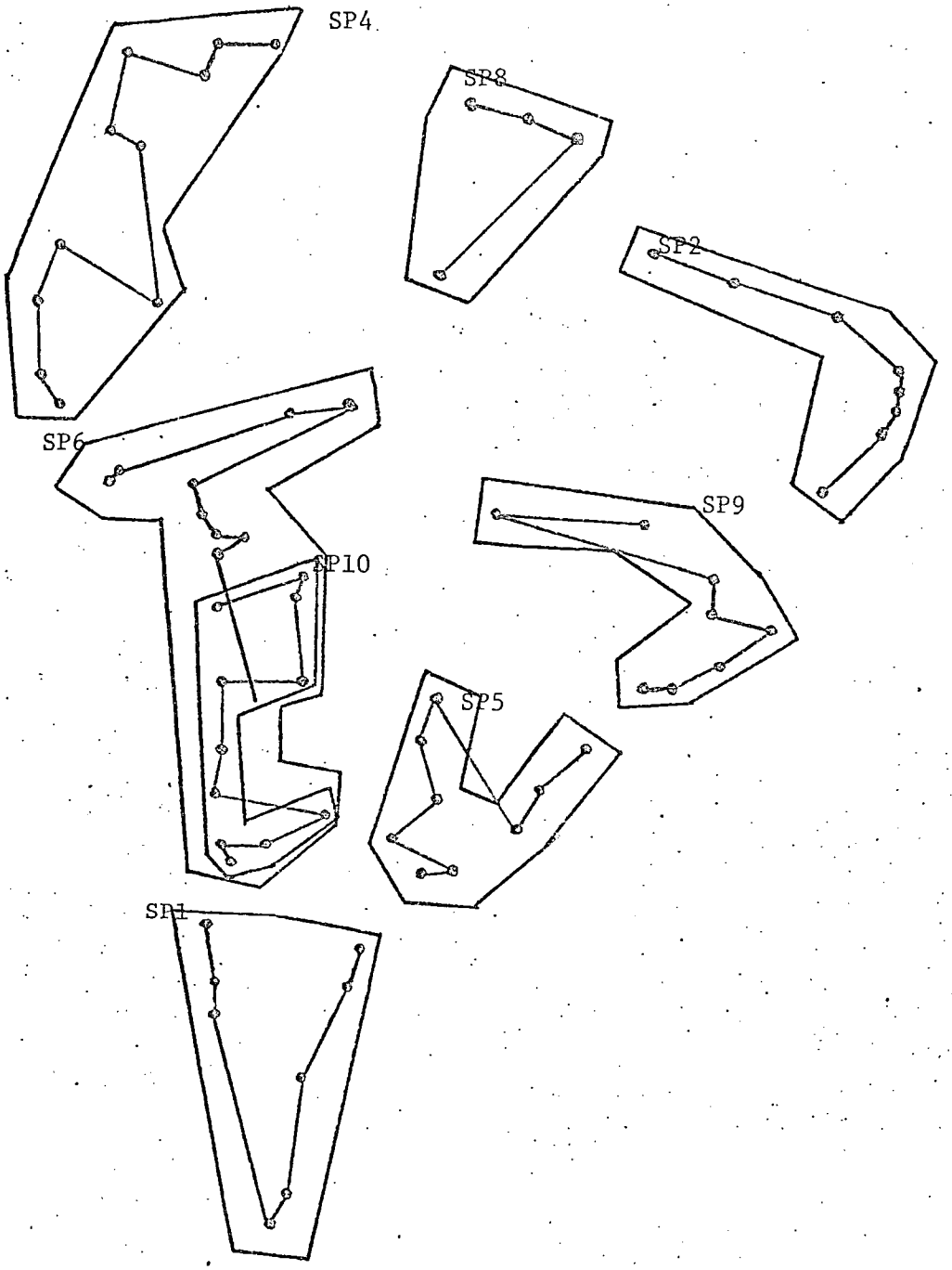


Figure 24: Solving All the Other Subproblems

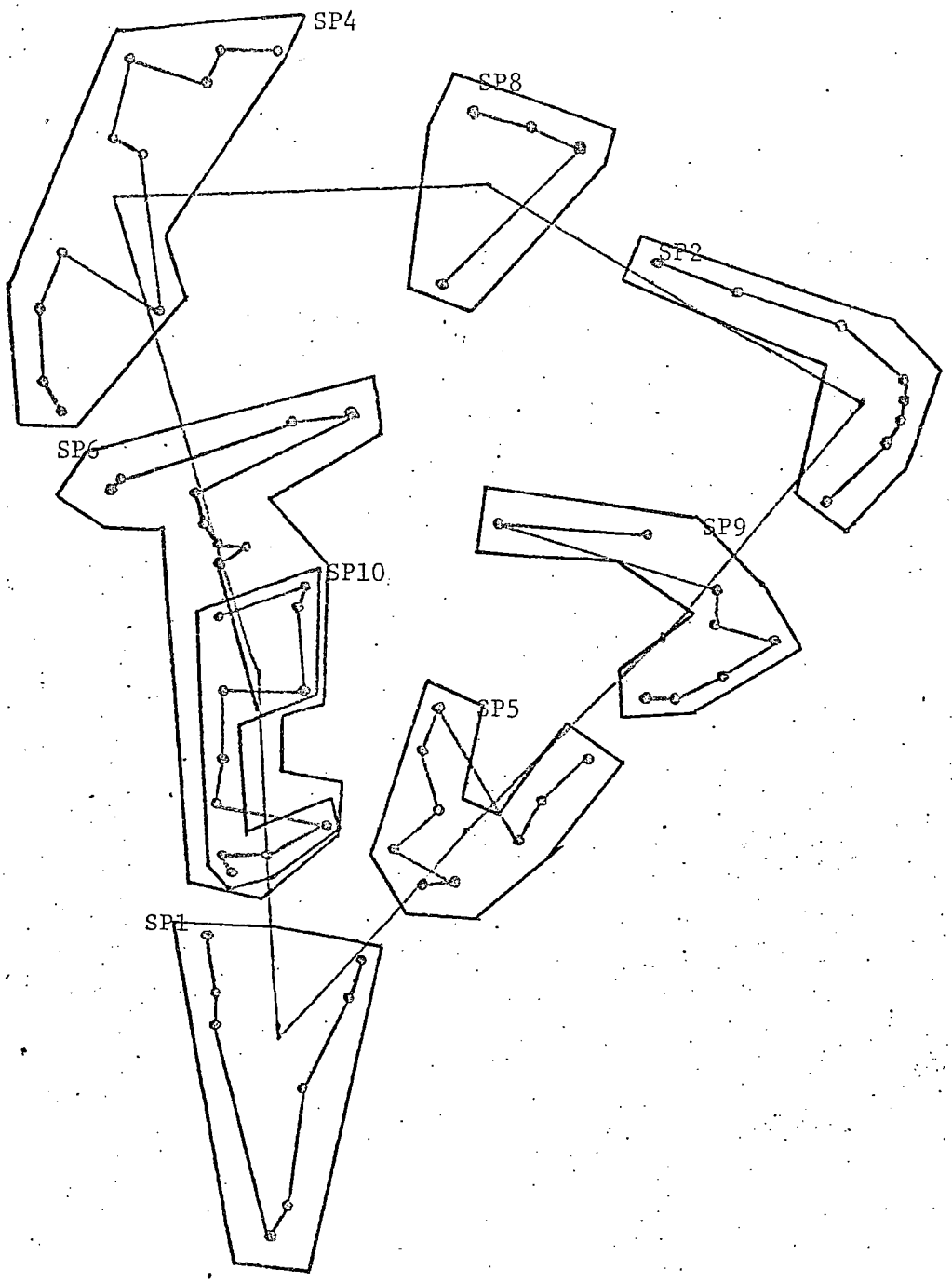


Figure 25: Solving the Top Level Subproblem

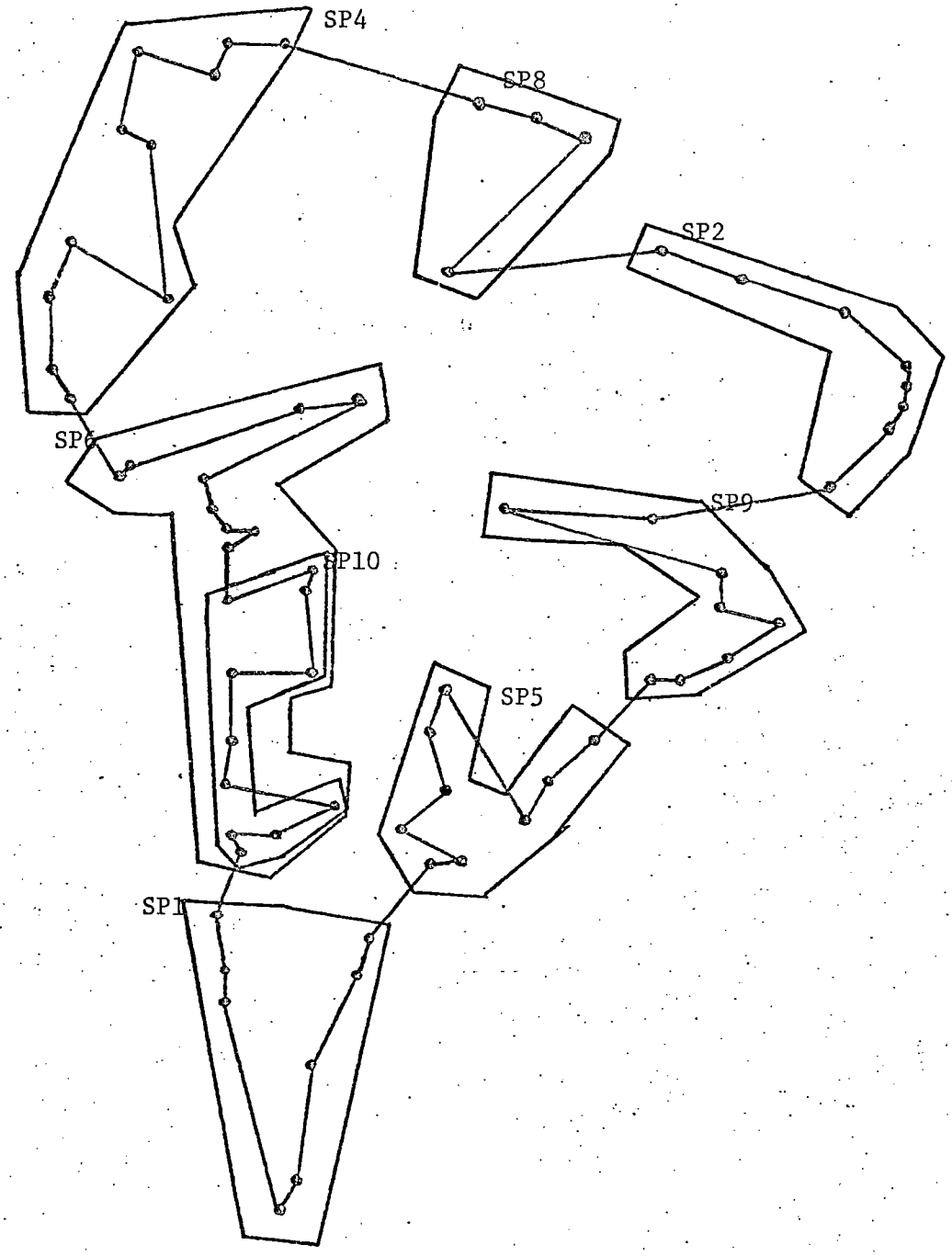


Figure 26: Synthesizing all the Subproblem Solutions

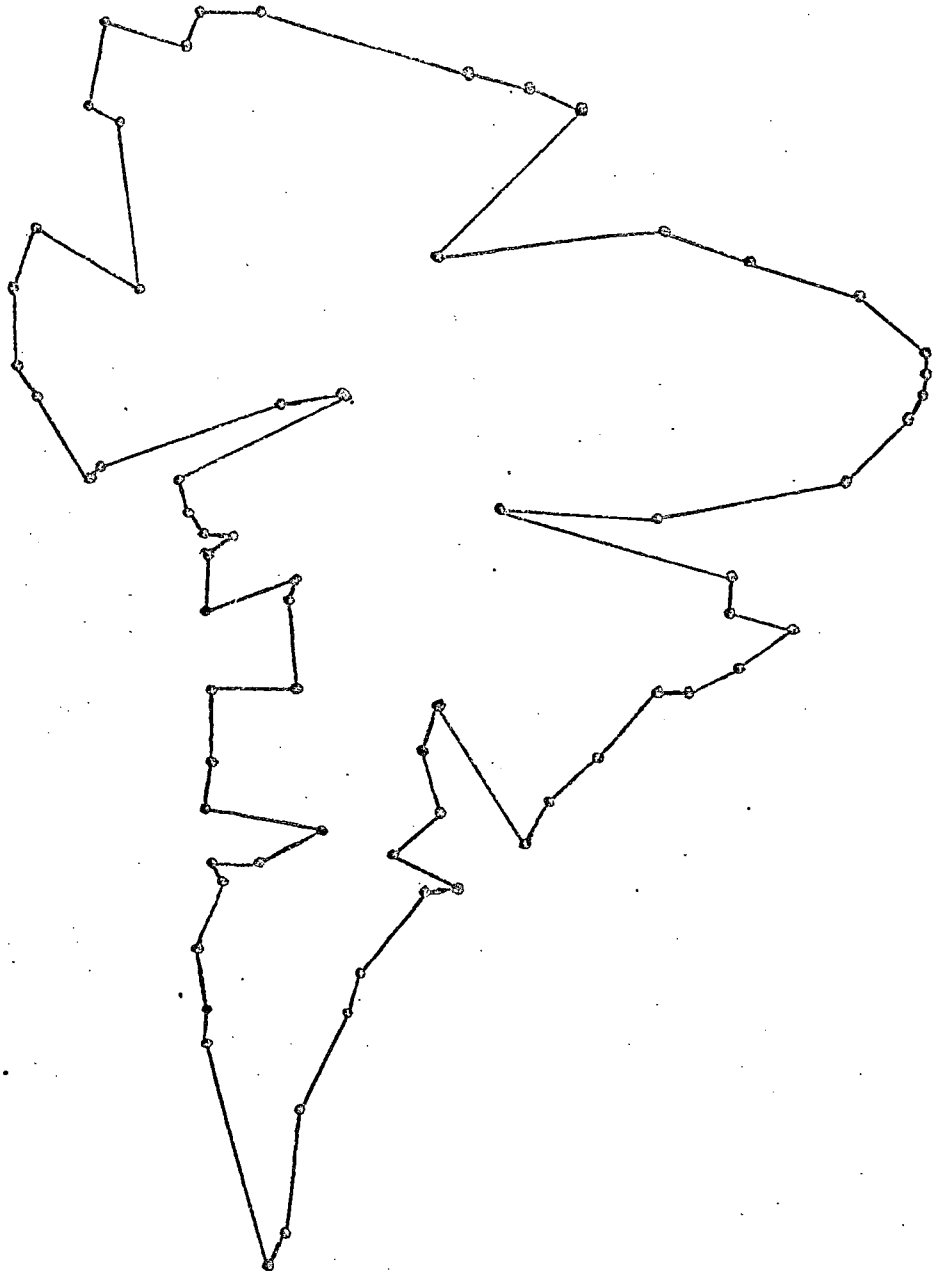


Figure 27:Erasing the Subproblem Names and Polygons

phase one. The contents of the new subproblems depend on the contents of the inner tour and the point at which it is to be joined to the outer tour. From figure 27 we decided there were three places where an inner tour might run very close to, and hence be cheaply joined to an outer tour: in the vicinities of cities 65, 66, and 67; cities 36, 37 and 38; and cities 41, 42 and 43 (figure 17). These are the dotted line areas in figure 28. Figure 28 is not a replica of a display picture.

Phase IIa We first experimented with some of the "joining" possibilities along the left hand side. This required the construction of a subproblem containing the "inner loop", one long subproblem on the right, and two subproblems on the left. The two on the left "break" at the point where the outer loop joins the inner loop.

We erased the synthesis solution and redisplayed the existing subproblems and subsolutions (figure 29).From figure 29 it was apparent that the following subproblems would have to be killed: SP4, SP5, SP6, SP8, SP9, and SP10. We decided at the same time to include the three points in a row in SP8 in SP2 so SP2 was killed, as well. This left us with a single remaining subproblem (figure 30).

We now created the subproblems SP11 and SP12 in figure 31. It was not clear whether city 23 (marked with a * in figure 31) should belong to the inner loop or the outer loop. Although we still had not decided where to make the break in the inner loop (i.e. SP14 and SP15 had not been constructed) we decided to resolve this question by comparing DIS(C4, C24) + DIS(C67,C23) + DIS(C68, C23) - DIS(C67,C68) with DIS(C4,C23) + DIS(C23,C24) + DIS(C67,C68) (see figure 17). The result of this comparison was an indication that C24 should be part of the inner loop. SP13 was then created (figure 31).

There were several ways to consider joining the inner loop with the left hand part of the outer loop. Our first inclination was to try the cheapest possible pair of joining links. Anticipating the join shown by

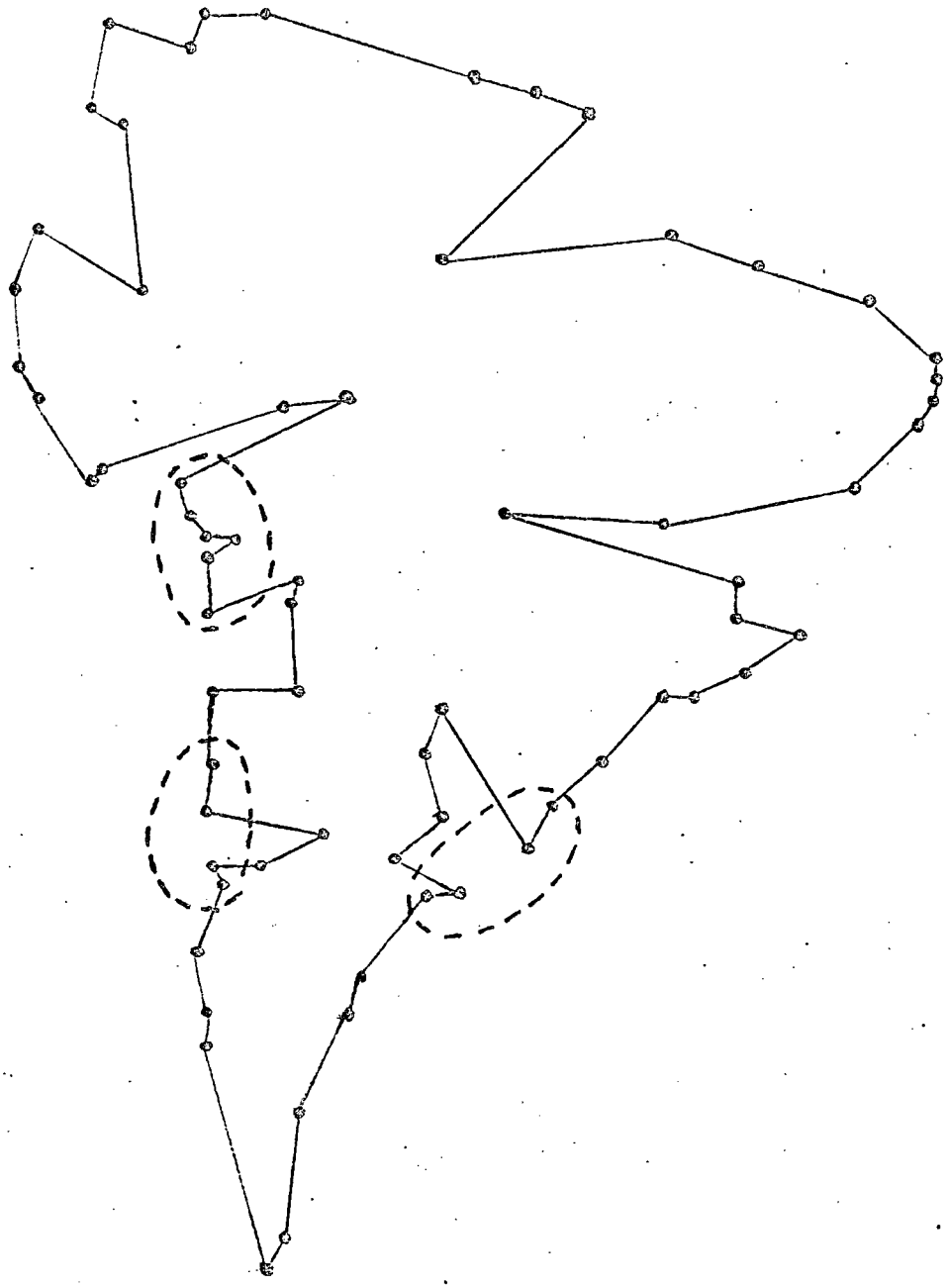


Figure 28: Joining Points for the Inner and Outer Tours

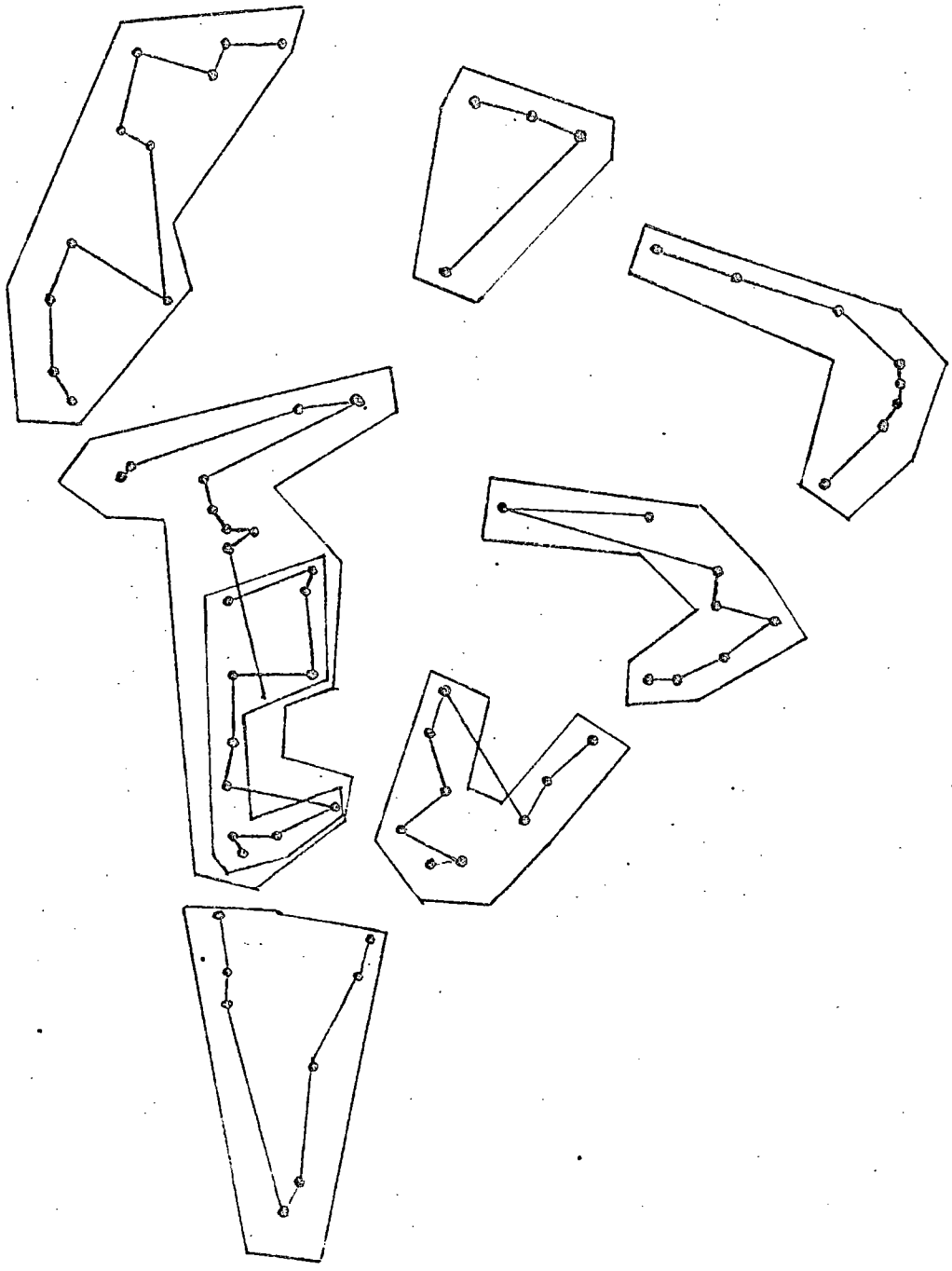


Figure 29: Existing Subproblems and Subsolutions

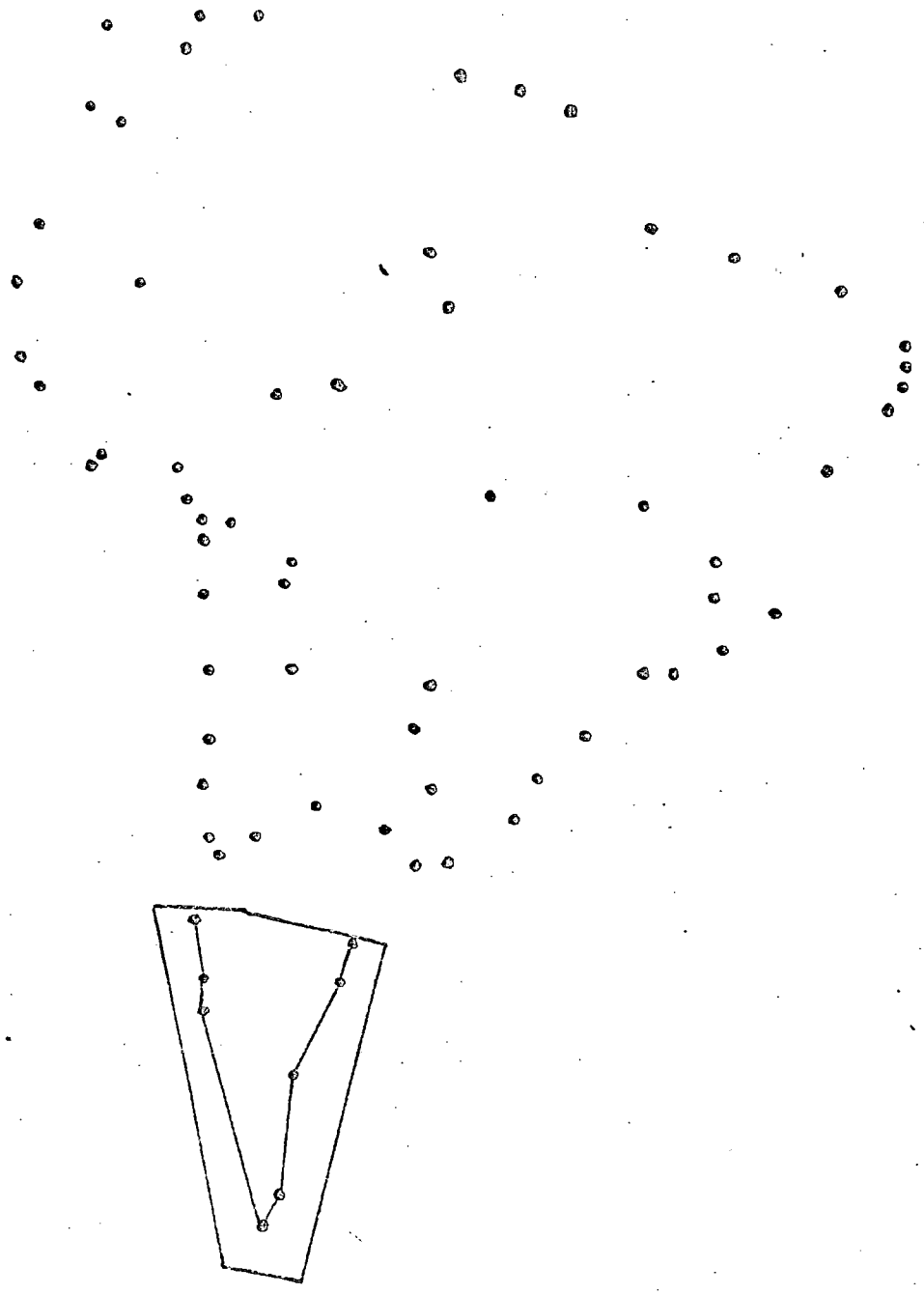


Figure 30: Remaining Subproblems and Subsolutions

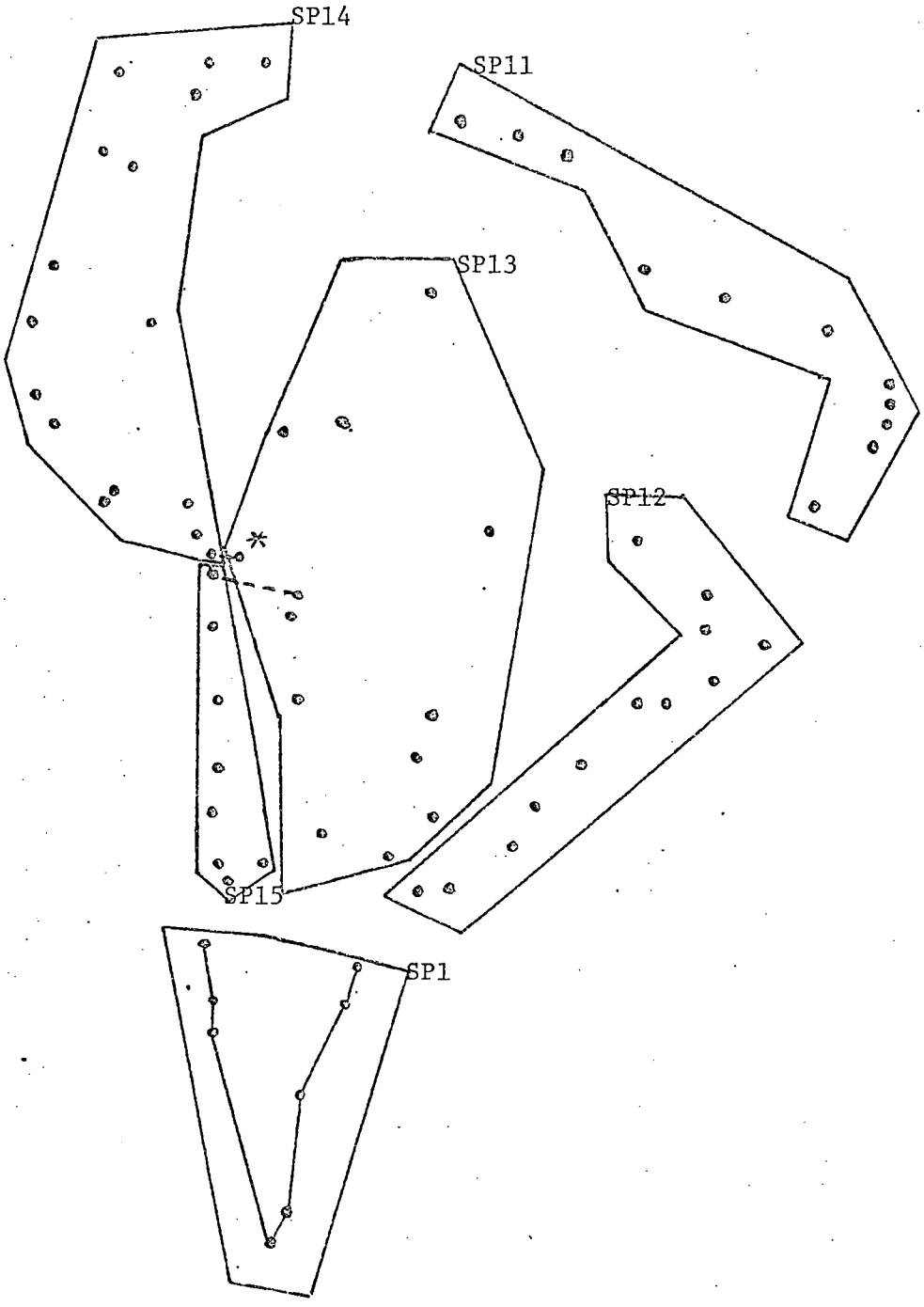


Figure 31: New Subproblems

the dotted lines in figure 30, we created subproblems SP14 and SP15. The subproblem names were then displayed and the result was (except for the dotted lines and *) the display in figure 31.

We now solved all of the subproblems. Because of our experience in Phase I we now had a good idea what the solutions should look like and which subproblem boundary points to choose. CLSOL was used to solve each subproblem and the resulting solutions were displayed (figure 32).

The top level subproblem was then solved, the subproblem solutions synthesized and all information but the synthesized solution erased (figure 33). The path length for this solution is 2749, a good improvement over the single tour solution from Phase I. The additional CPU time required was 117.5 seconds and the "sitting" time about 15 minutes.

Phase IIb We examined our Phase Ia solution to see if we could see any way to further reduce its cost. The variable aspect of the Phase II idea is the choice of the pair of links required to join the two circular tours. After a little thought it was evident that we should try to minimize the inter-loop links but maximize the "breaks" in the inner and outer loops where they are joined. The joining links in the IIa solution are small but the tour breaks at the joining places are not very large. There are several places at which a large possible break in the inner loop is opposite a large possible break in the outer loop. There are two such obvious places on the left side of the inner tour and one on the right.

The process of trying out these ideas involves killing and creating subproblems, re-establishing a top level solution and then creating a synthesis. The process is the same as that for creating the Phase IIa solution and will not be described here. The three additional synthesized solutions resulting from three alternative choices of ways of joining the inner and outer loops are shown in figures 34, 35 and 36. The path length for these three solutions are 2733, 2735, and 2741. The additional CPU time required to create these three solutions was 22, 20 and 40 seconds. The sitting and

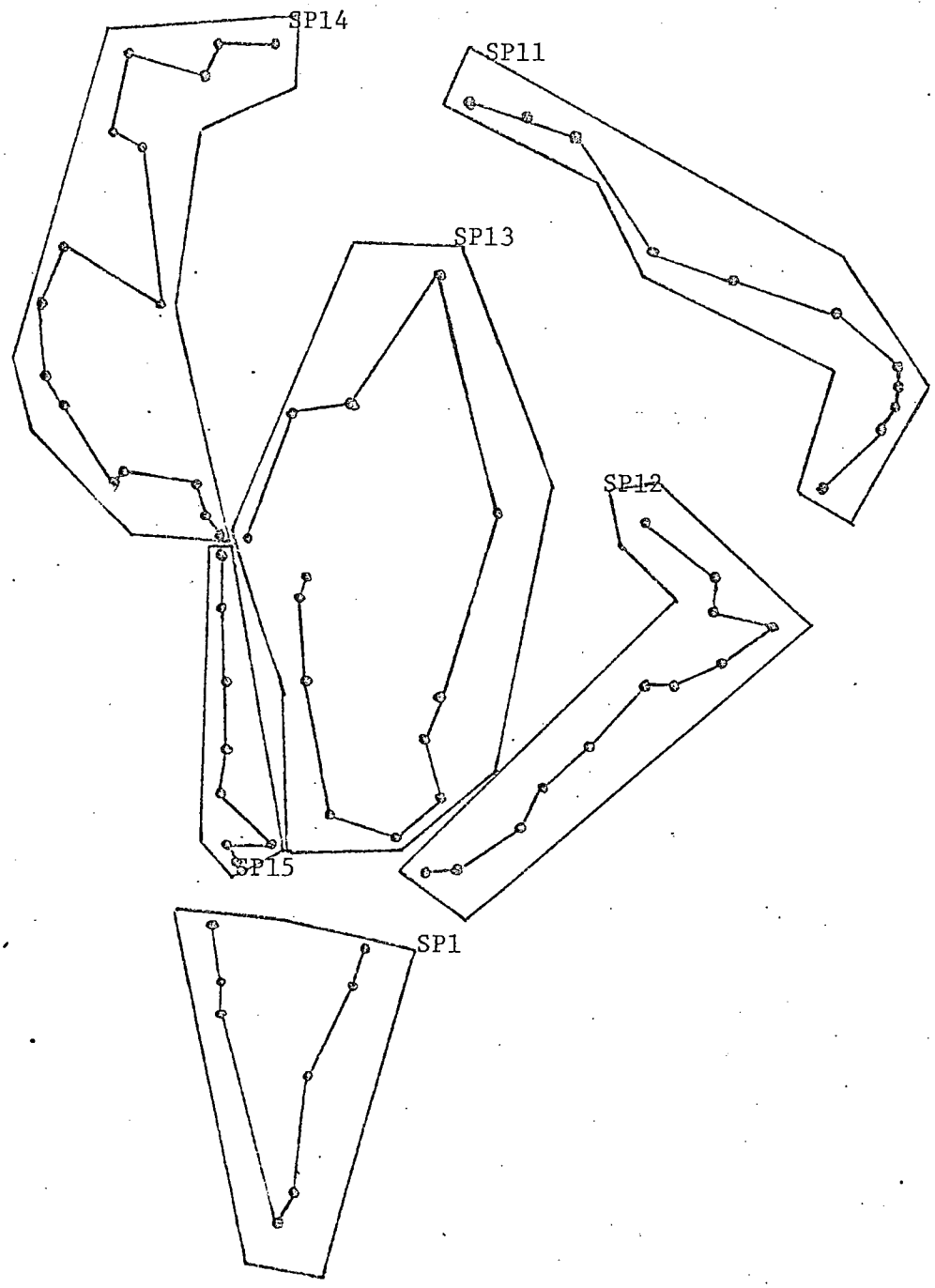


Figure 32: New Subproblem Solutions

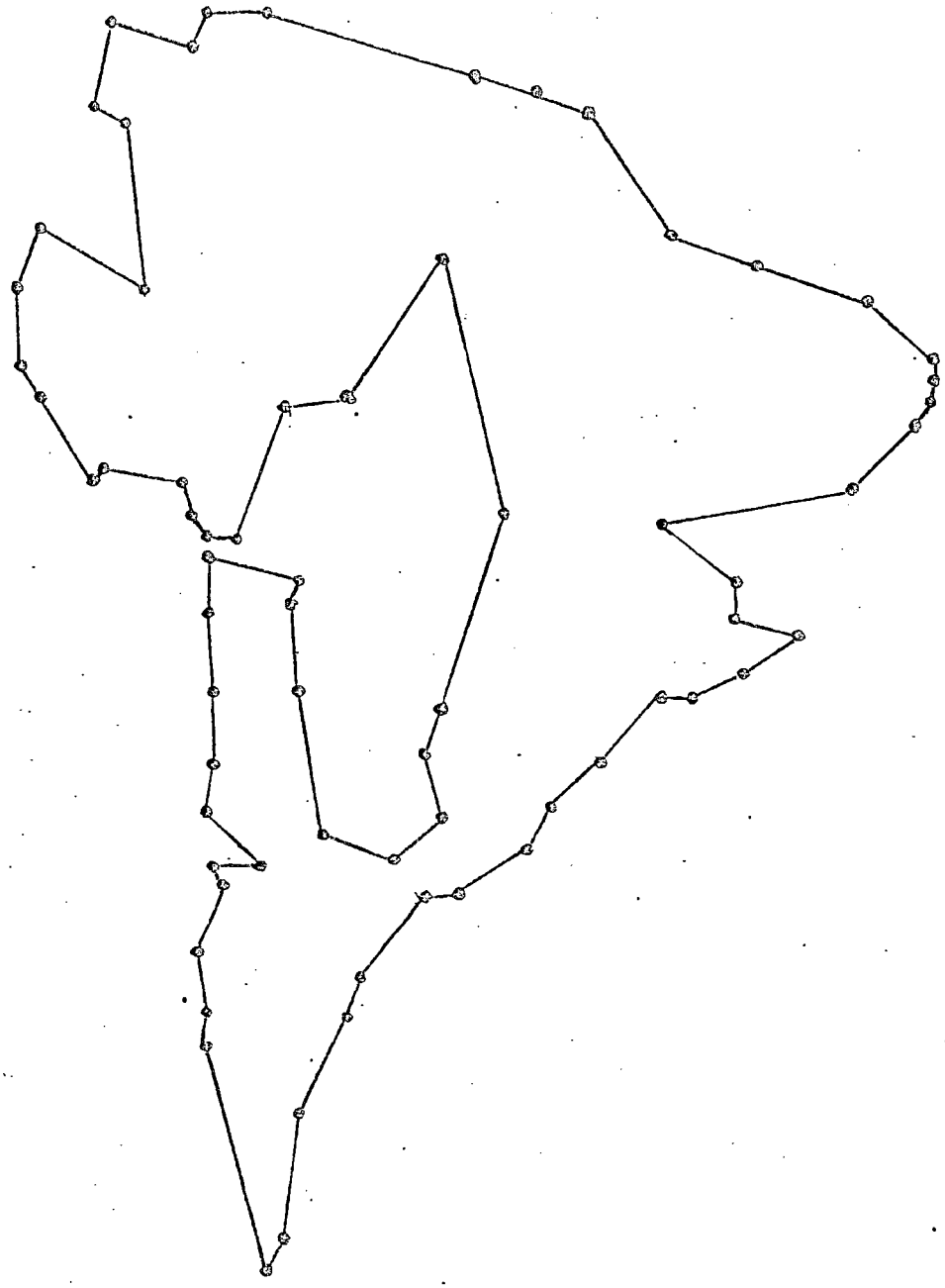


Figure 33: Synthesized Solution

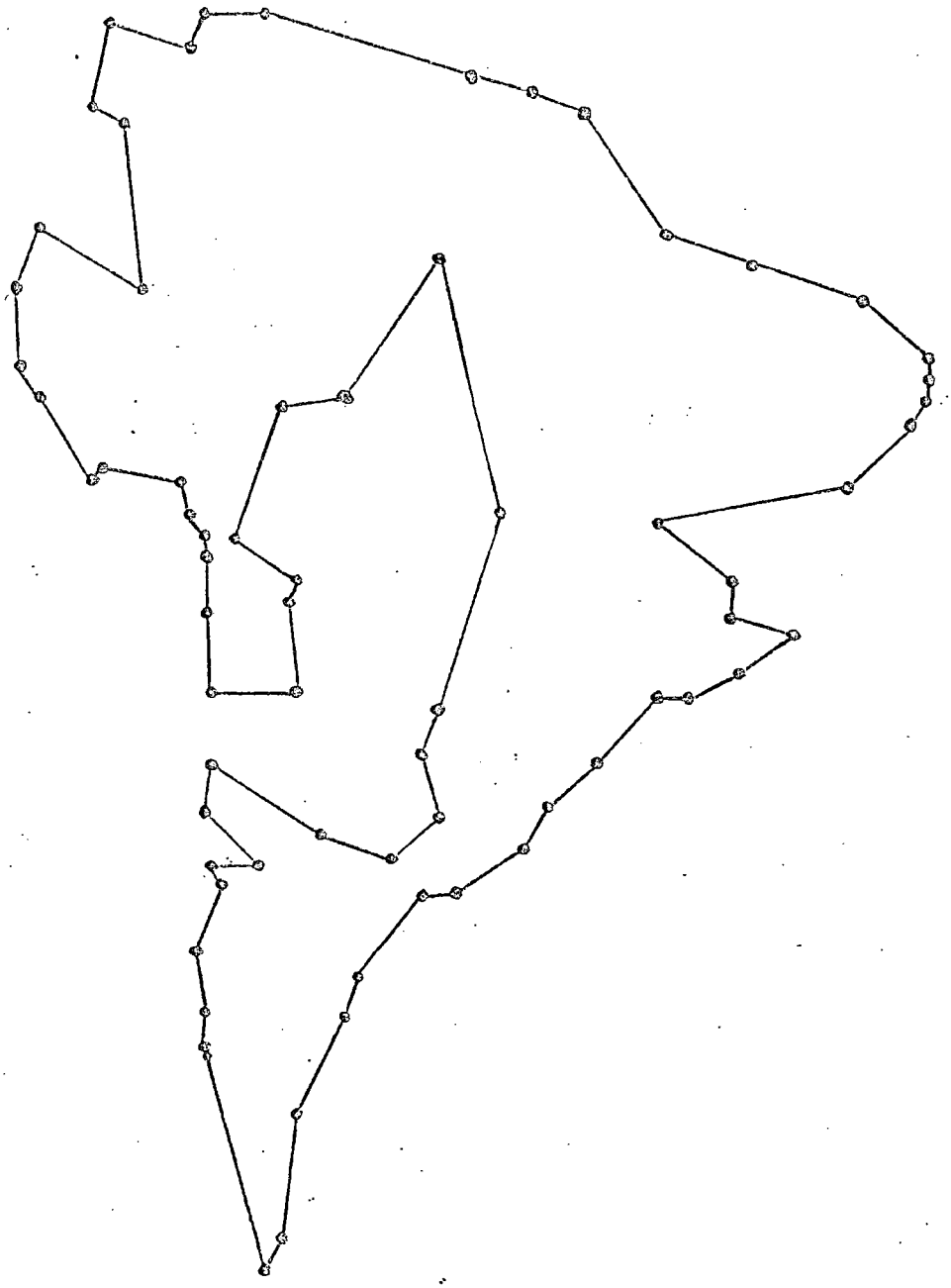


Figure 34: Solution with Path Length 2733

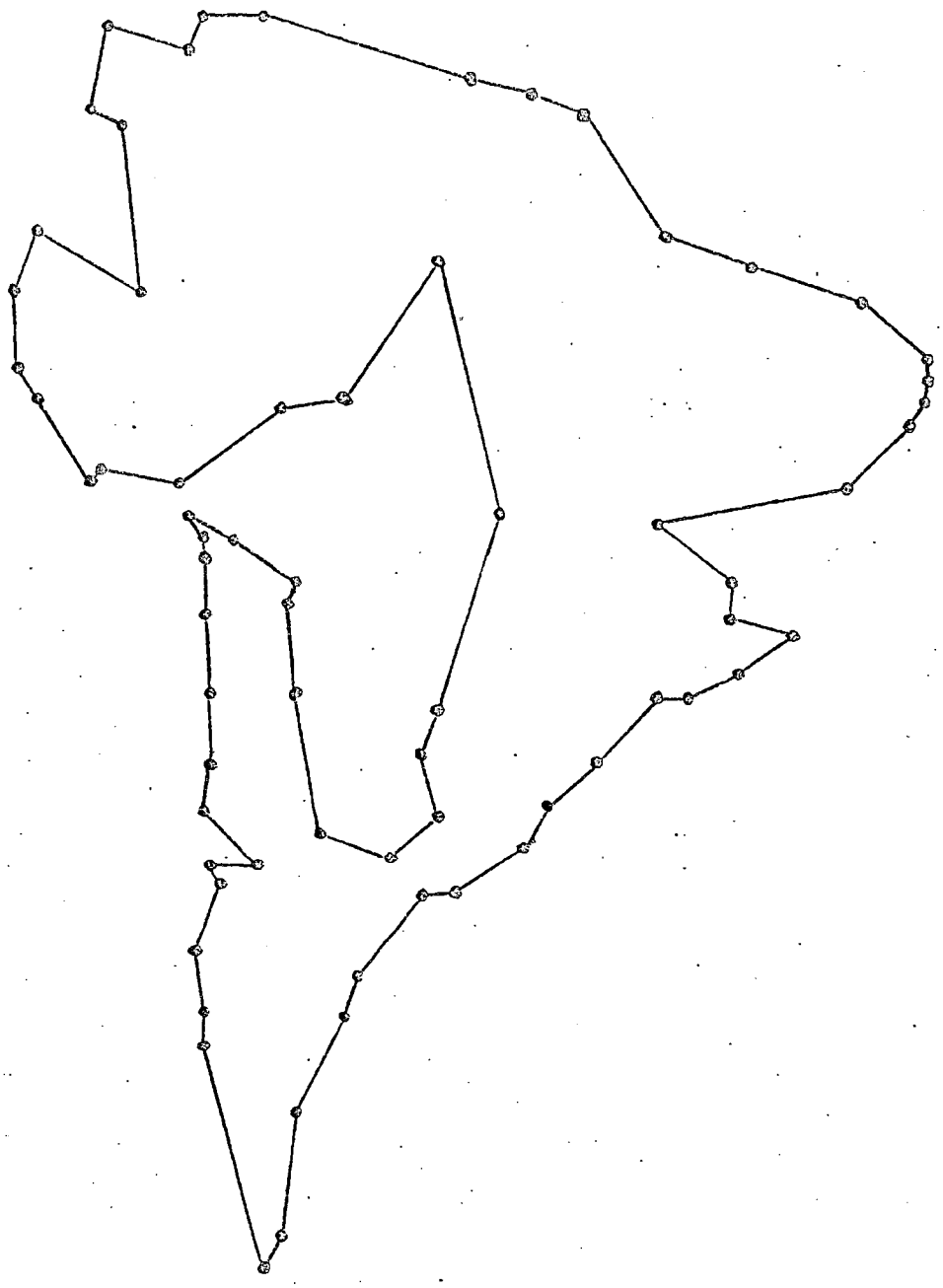


Figure 35: Solution with Path Length 2735

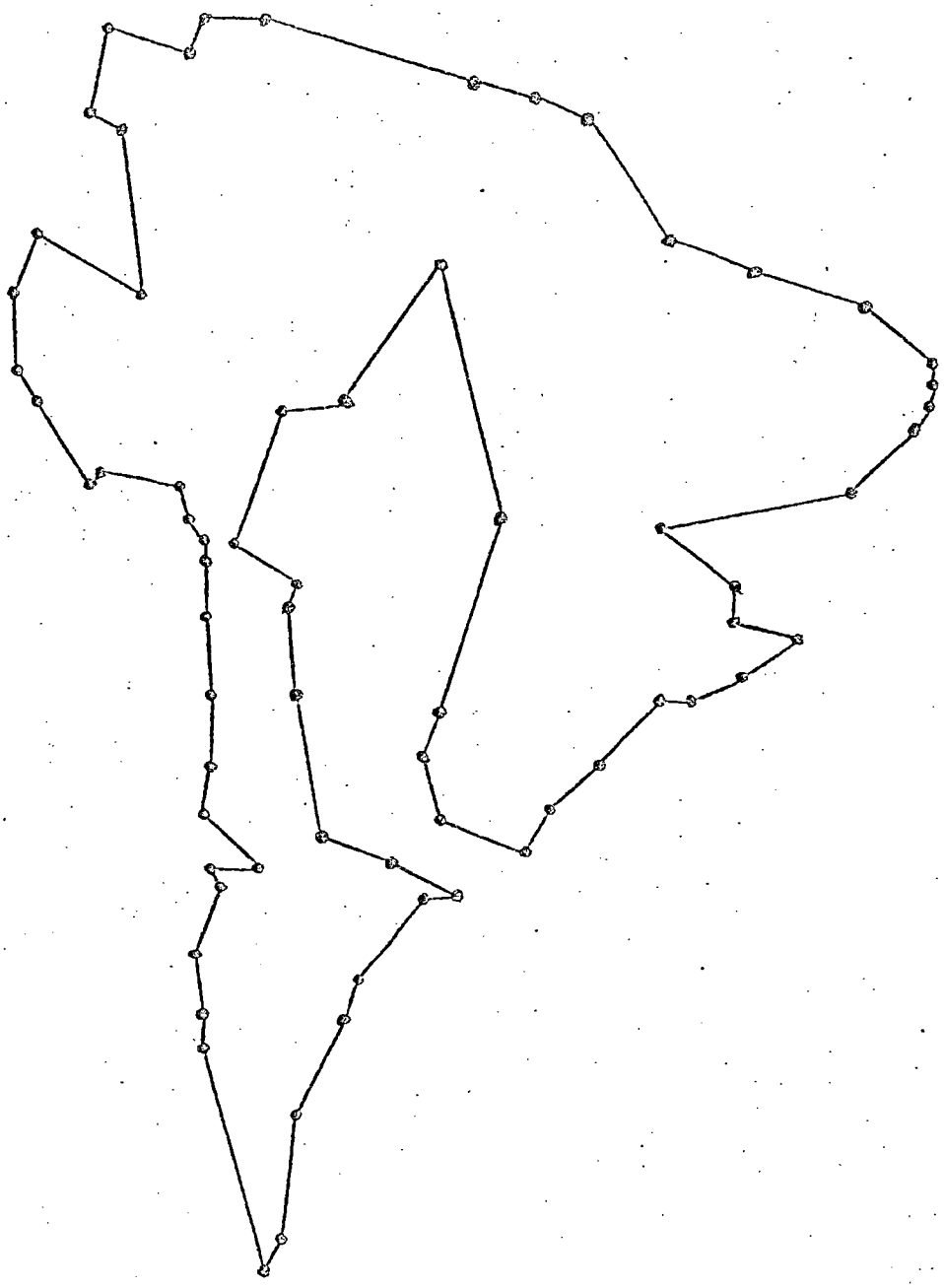


Figure 36: Solution with Path Length 2741

thinking time was less than 10 minutes in each case.

Phase III In this phase, the entire problem was treated as a single subproblem and solved using first CCSOL and then CLSOL. Recall that CKSOL can not be used for problems with more than 12 cities. The South American problem has 68 cities.

The resulting Croes solution (figure 37) (CCSOL) had a path length of 3219. Its computation required 540 seconds CPU time. The procedure was run with the default setting of two reduction cycles and two 2-opt tours per reduction cycle.

The Lin solution (figure 38) had a path length of 2741 and required 1296 seconds CPU time. CLSOL was also run with the standard default setting.

10. The France, Spain and Italy Travelling Salesman Problem

The account of the solution of this problem is more condensed than that provided for the South American Problem. The details of the solution process are the same as for that problem. We were more confident about the use of the system for this problem and proceded more rapidly in the construction of solutions. We had also gained a confidence in the CLSOL subproblem solver and used it throughout this solution process. Three general solution ideas occurred to us in the solution process.

Phase I We first decided to try the obvious grouping of the cities into Spain, France and Italy. These subproblems were constructed and their subproblem names displayed, (figure 40). Subsolution boundary points were chosen and CLSOL called (figure 41). In SP2 there were two simple choices for the endpoint which would interface with SP3. We computed a solution to

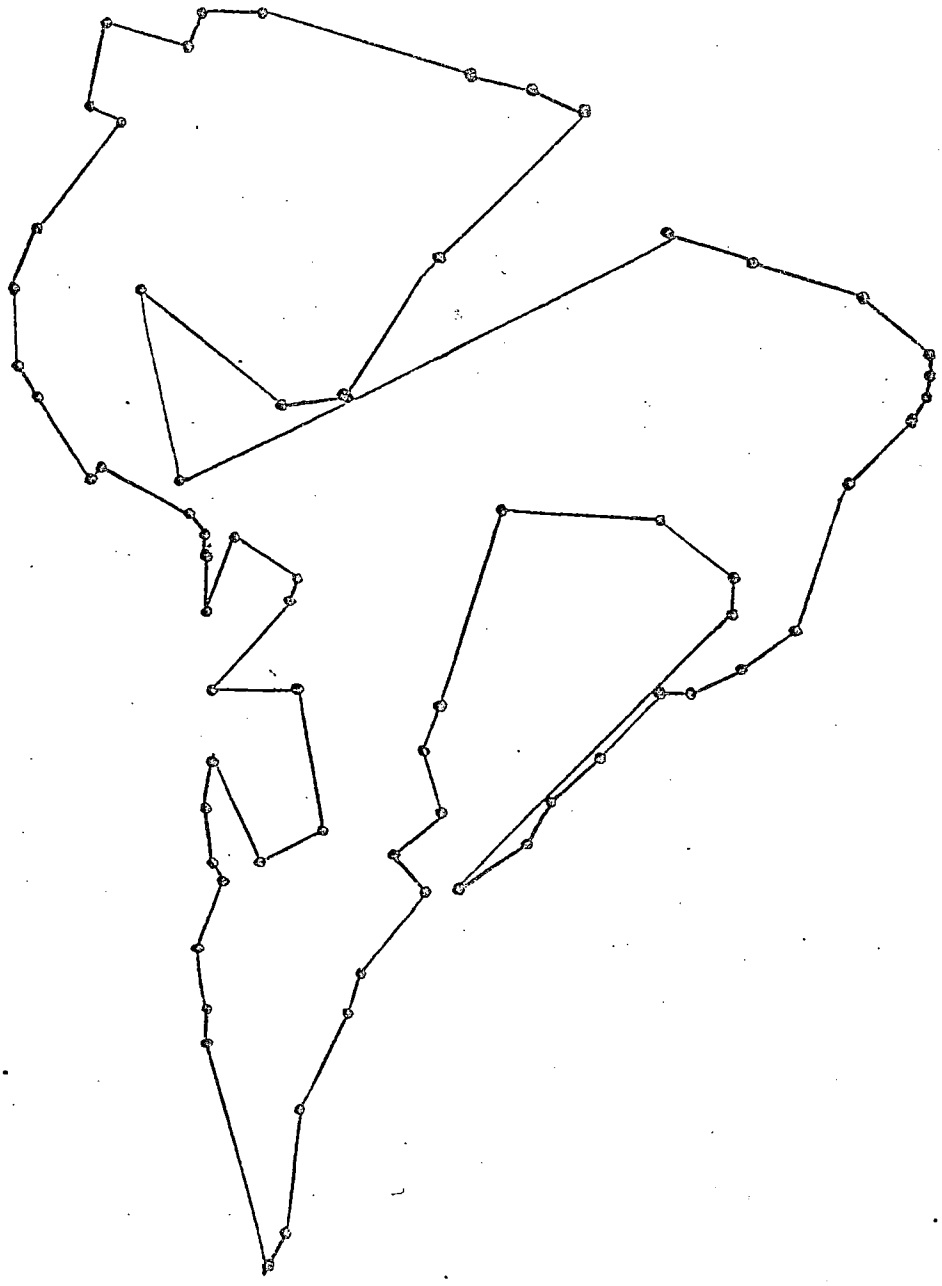


Figure 37: The Croes Solution

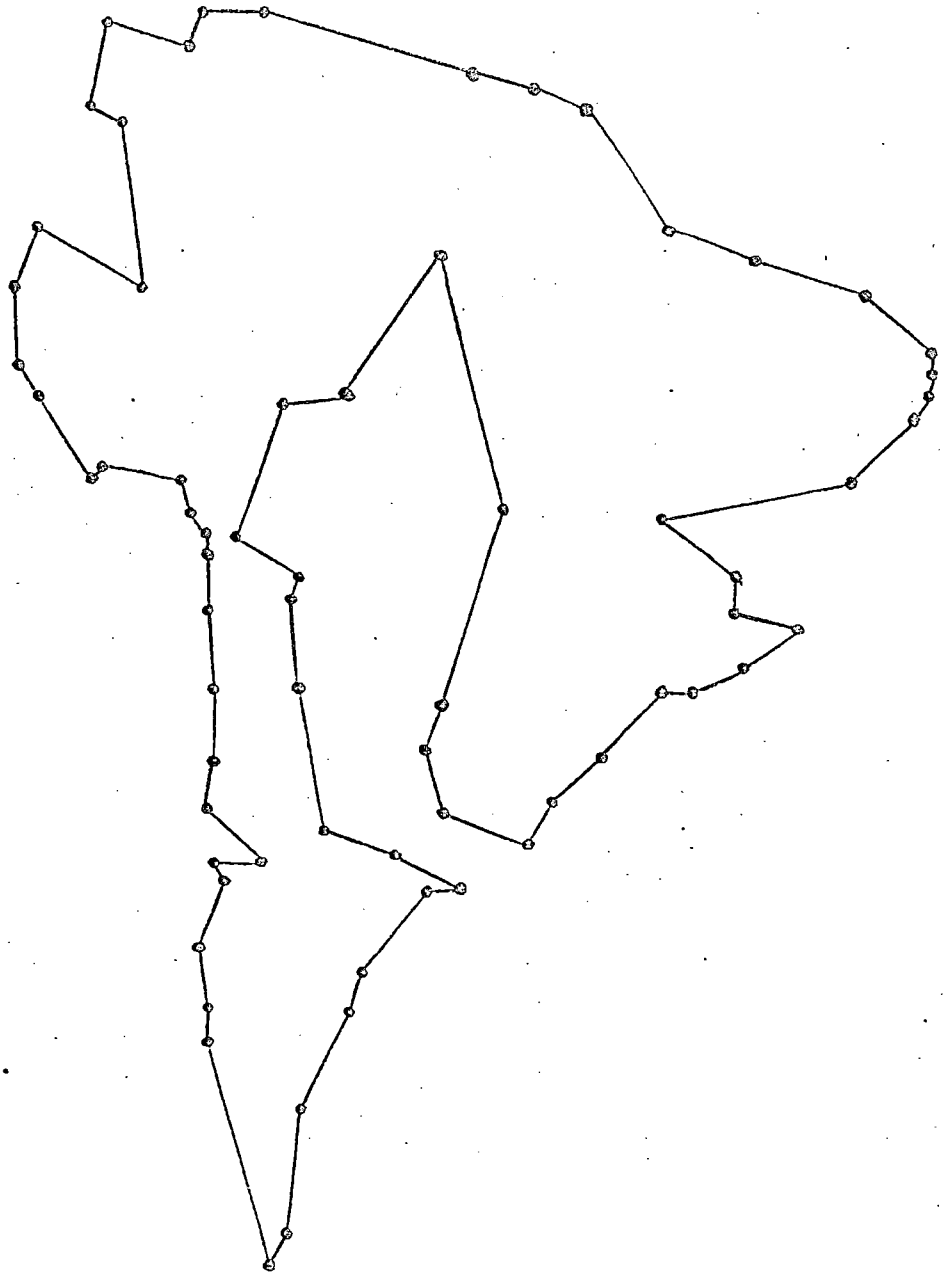


Figure 38: The Lin Solution

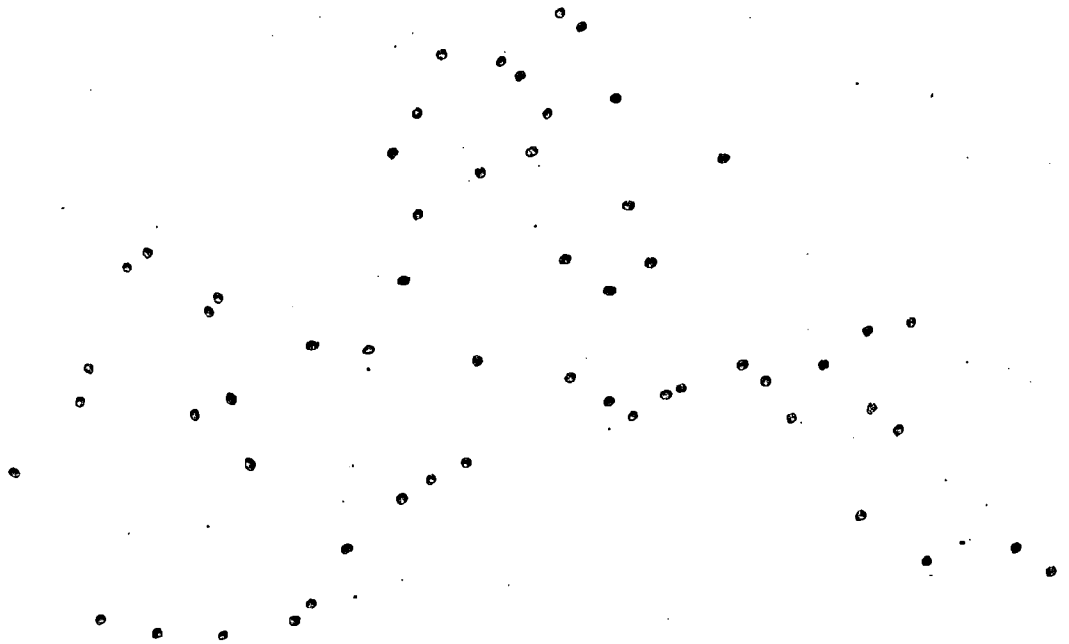


Figure 39: The France, Spain and Italy Travelling Salesman Problem

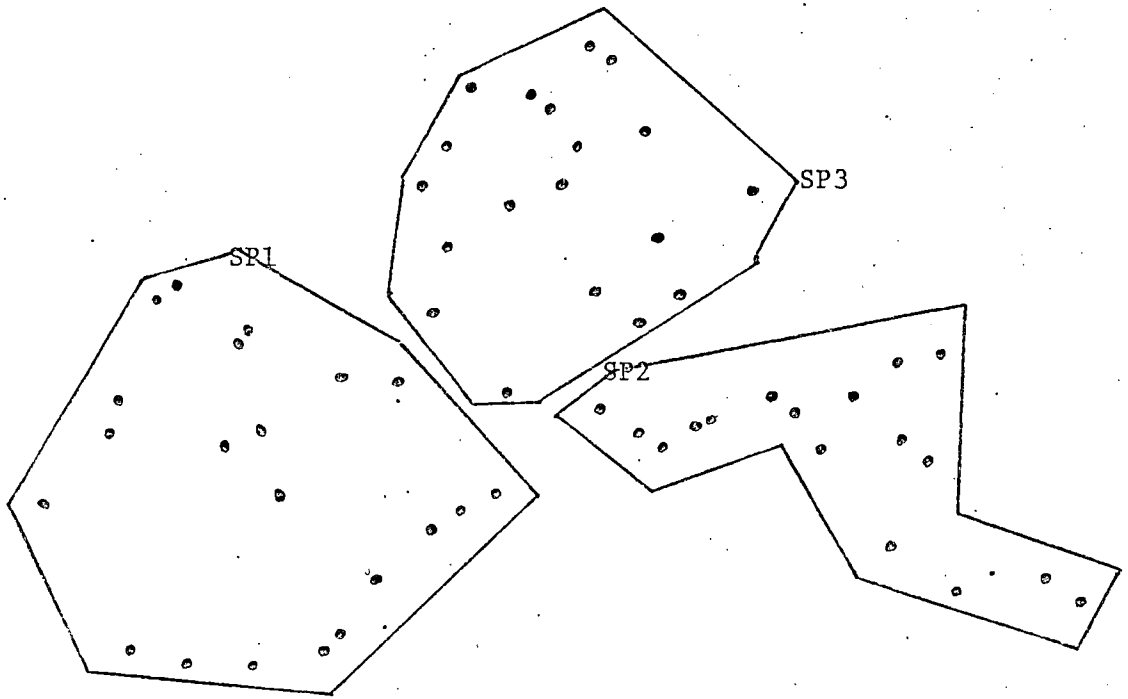


Figure 40: Creation of the Obvious Subproblems

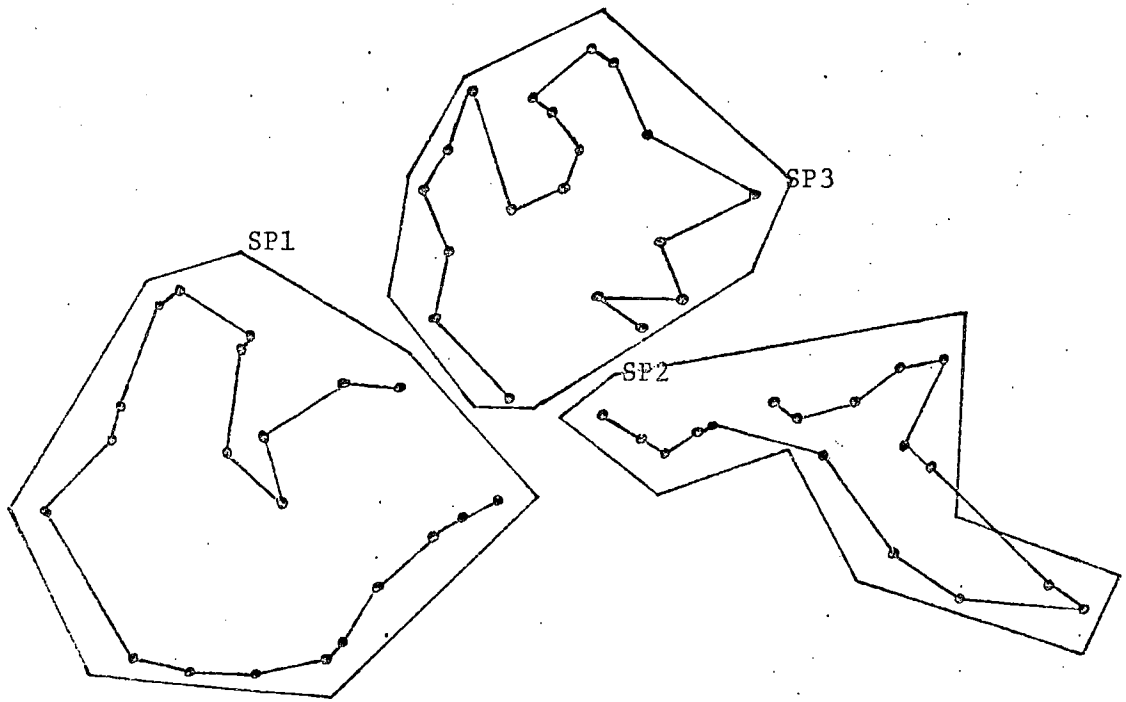


Figure 41: Subproblem Solutions

SP2 for both choices and added in the lengths of the associated interface links in deciding on the choice in figure 41.

The top level subproblem was then solved, the solutions synthesized, and the subproblem names and polygons erased (figure 42). The path length was 2094, the elapsed time 15 minutes and the CPU usage 40 seconds.

We now chose to recompute the solution with two changes. The first was to use a different endpoint in SP3 for linking with the upper endpoint in SP2. The second was to include the lower endpoint in SP2 in SP3. The idea was that it would be cheaper to visit that city on the way to SP2 from SP1 rather than SP3. The new subproblems and subproblem solutions are shown in figure 43. The synthesized solution is shown in figure 44. The path length for this solution is 2047. The CPU usage required was 40 seconds. The elapsed time was 10 minutes.

Phase II Recalling our experience with the inner tour approach in the South American problem we decided to try joining the inner group of cities in Spain and France into a separate tour. There wasn't much we could think of to do for Italy. The appropriate subproblems and endpoints were chosen and the subproblems chosen, resulting in the solution structure in figure 45.

The synthesized solution (figure 46) required 15 minutes elapsed time and 68 CPU seconds. The path length was 2120, invalidating the big inner loop idea for this problem.

Phase III Even though the bigger inner loop idea was no good, we thought that perhaps a small inner loop in France (SP5 in figure 43) might work. In thinking of how we would construct a subproblem to force the inner loop, we realized that if the inner loop idea was any good, such a loop would

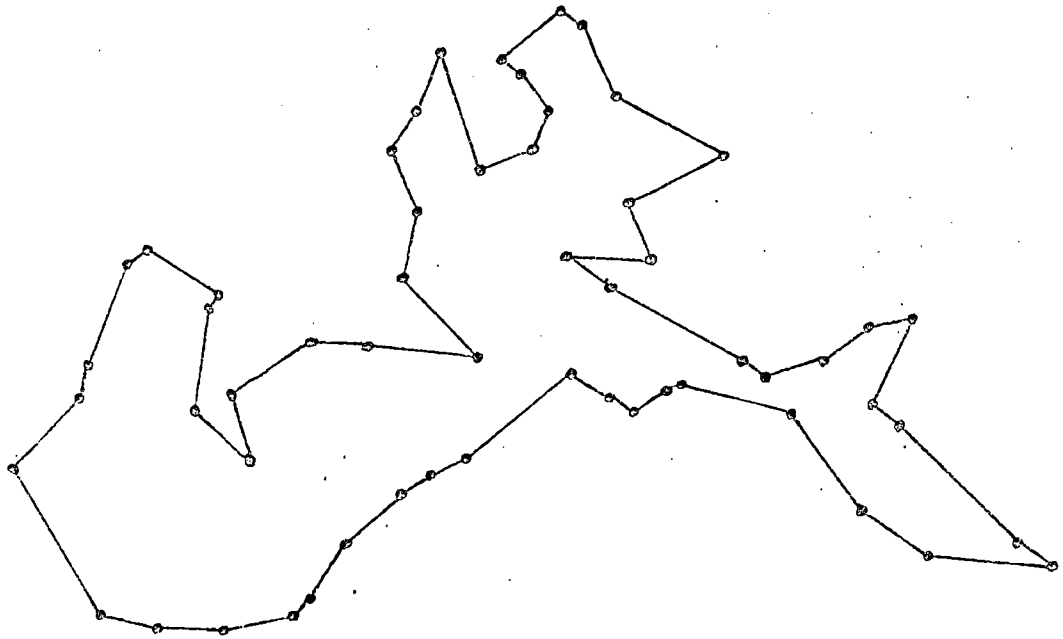


Figure 42: Synthesized Solution with Path Length 2094

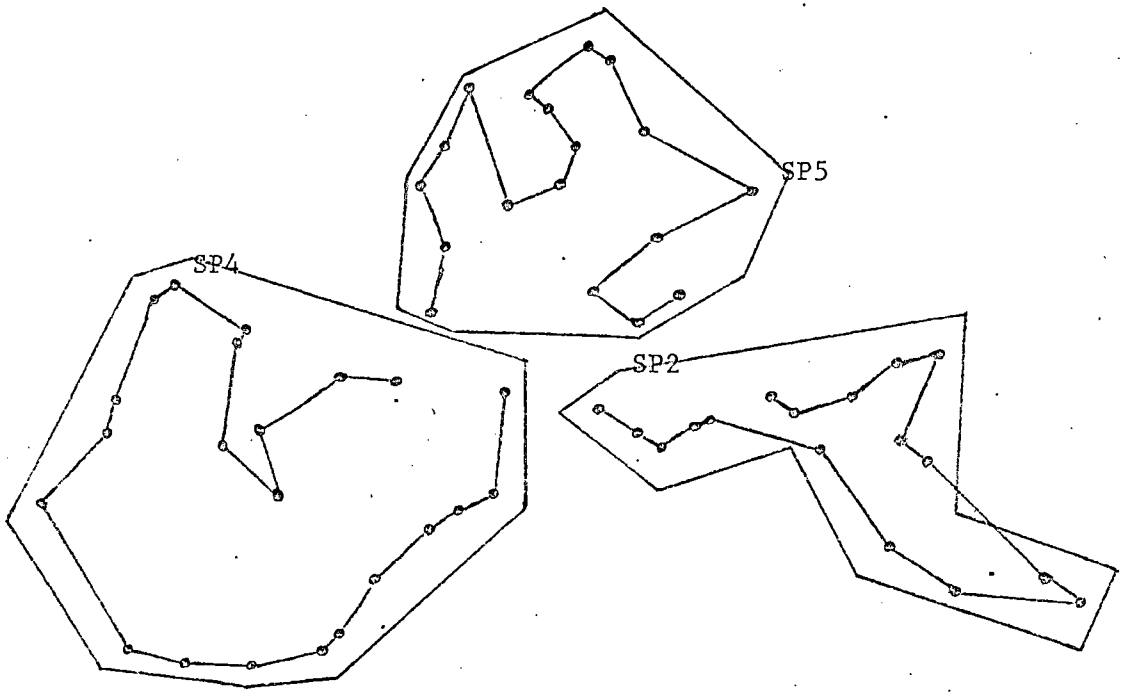


Figure 43: New Subproblems and Solutions

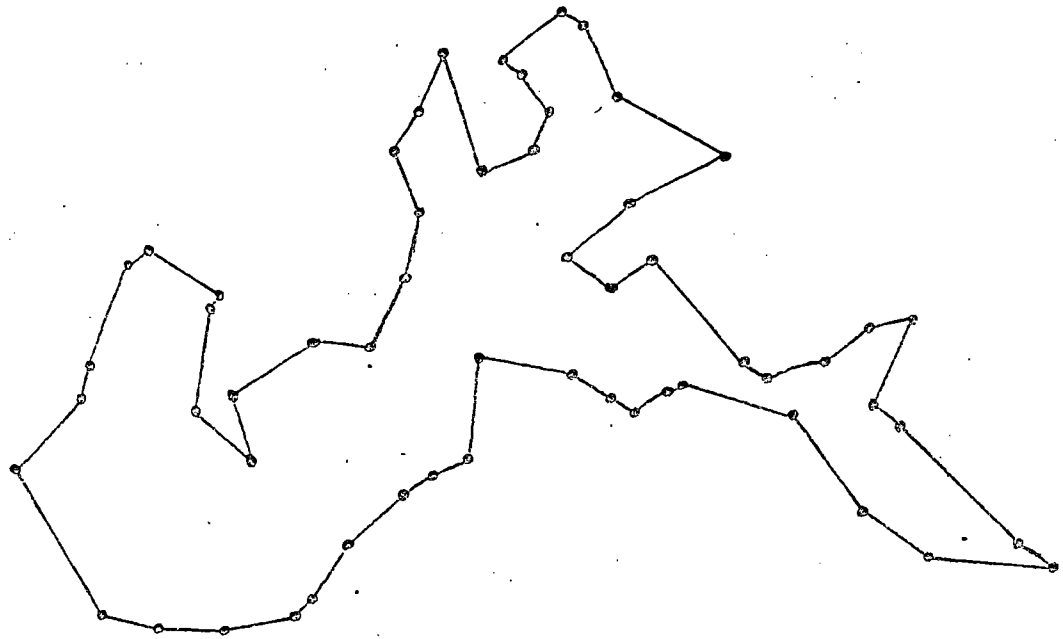


Figure 44: Synthesized Solution with Path Length 2047

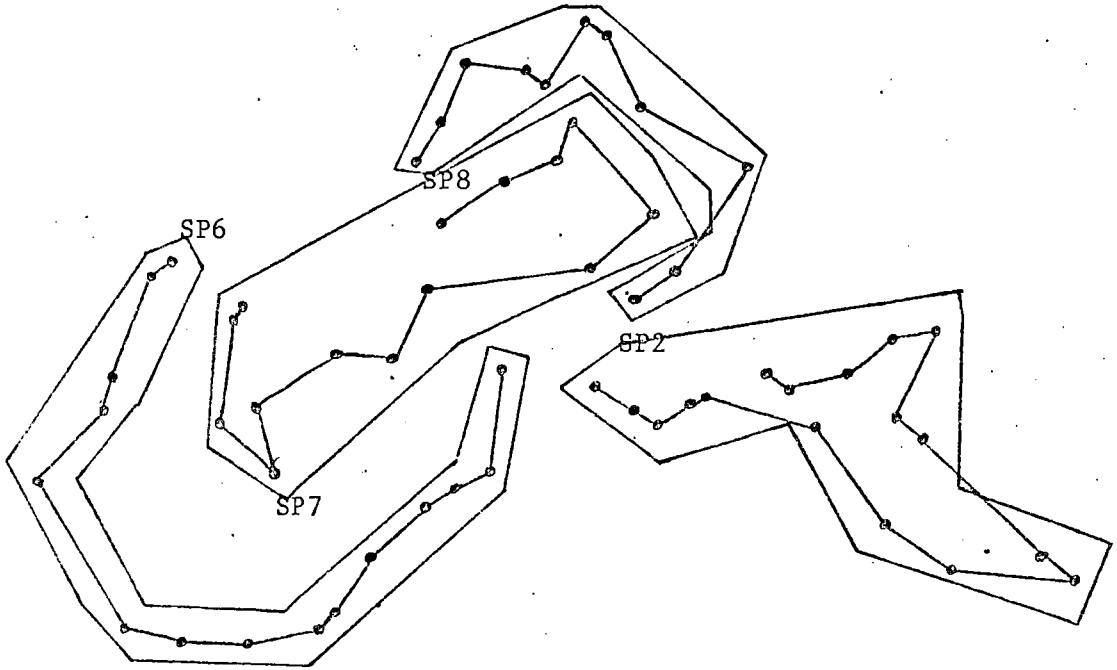


Figure 45: Suproblems and Solutions for the Big Inner Loop Solution Idea

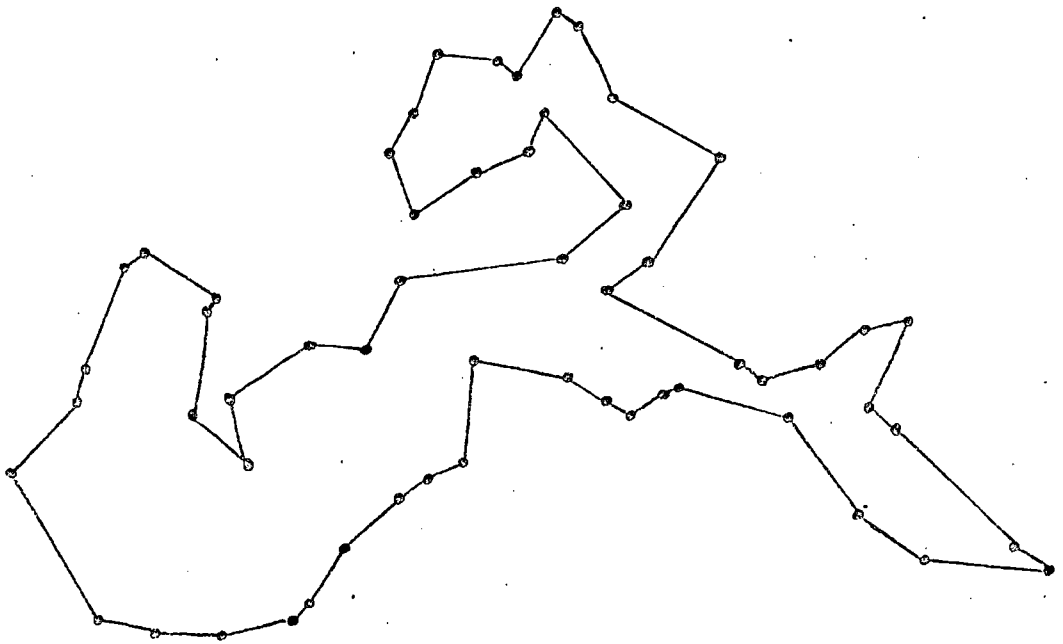


Figure 46: Synthesized Solution with Path Length 2120

occur automatically during the solution of SP5. We recreated the three subproblems in figure 43 and the solution to SP4. Imagining how such an inner loop in SP5 might run, we chose the appropriate endpoints and solved SP5. The result is shown in figure 47.

A synthesis was computed. The resulting solution (figure 48) required 60 CPU seconds and an elapsed time of 12 minutes. The path length is 2034, the best solution to this problem so far.

The only obvious improvement to this solution was to try a different upper endpoint in SP2 for the link to SP9. We re-solved SP2 with the new endpoint resulting in the synthesized solution in figure 49. The additional elapsed time was 5 minutes and the CPU cost 11 seconds. The new path length was 2021.

An examination of the synthesis in figure 49 revealed one more slight possibility for improvement by another change in the upper endpoint of SP2. The resulting synthesized solution (figure 50) required 9 additional CPU seconds and 4 elapsed minutes. The resulting solution had a path length of 2012.

Phase IV The problem was then solved automatically by treating it as a single subproblem. CCSOL produced a solution with path length 2140 in 169 seconds (figure 51). CLSOL produced the same solution as the final interactive solution in 814 CPU seconds.

11. The Eire Travelling Salesman Problem.

In section 8 we mentioned that we chose the third of these three problems because it apparently had little subproblem structure. The cities for this problem are uniformly distributed (figure 52).

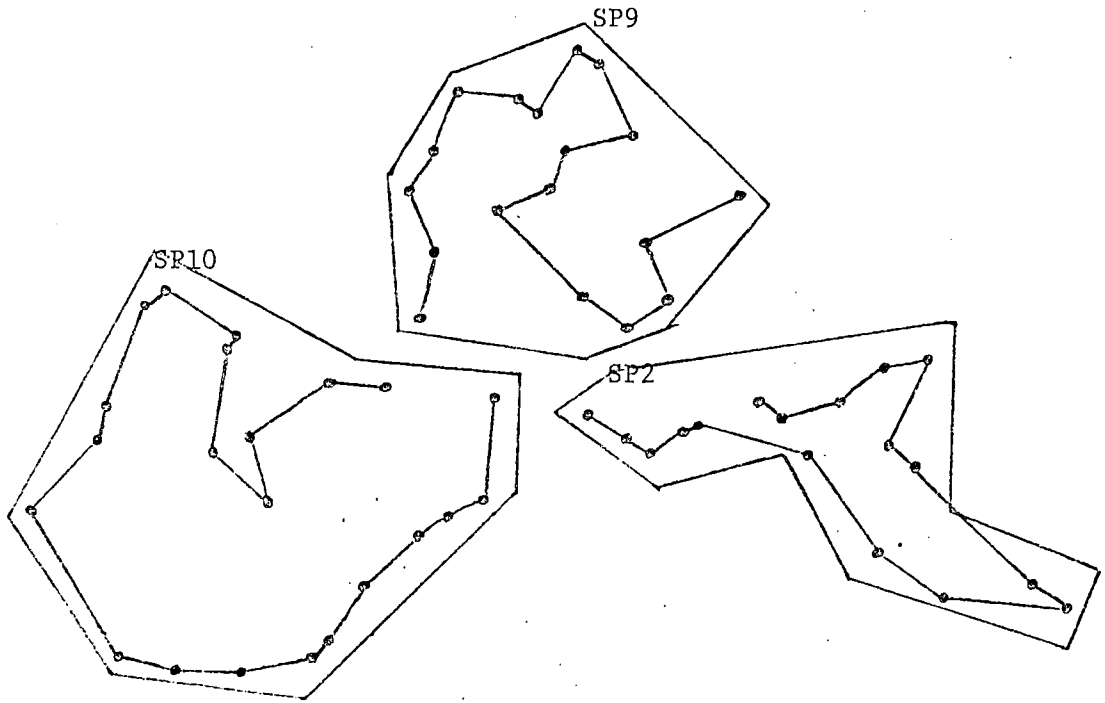


Figure 47: Inner Loop Solution to SP9

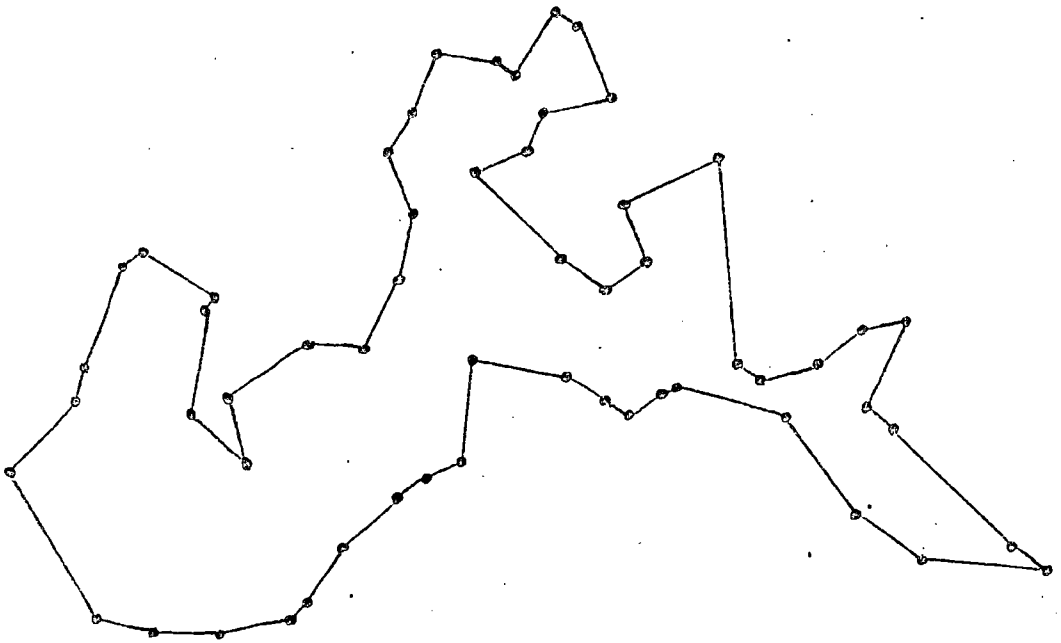


Figure 48: Synthesized Solution with Path Length 2034

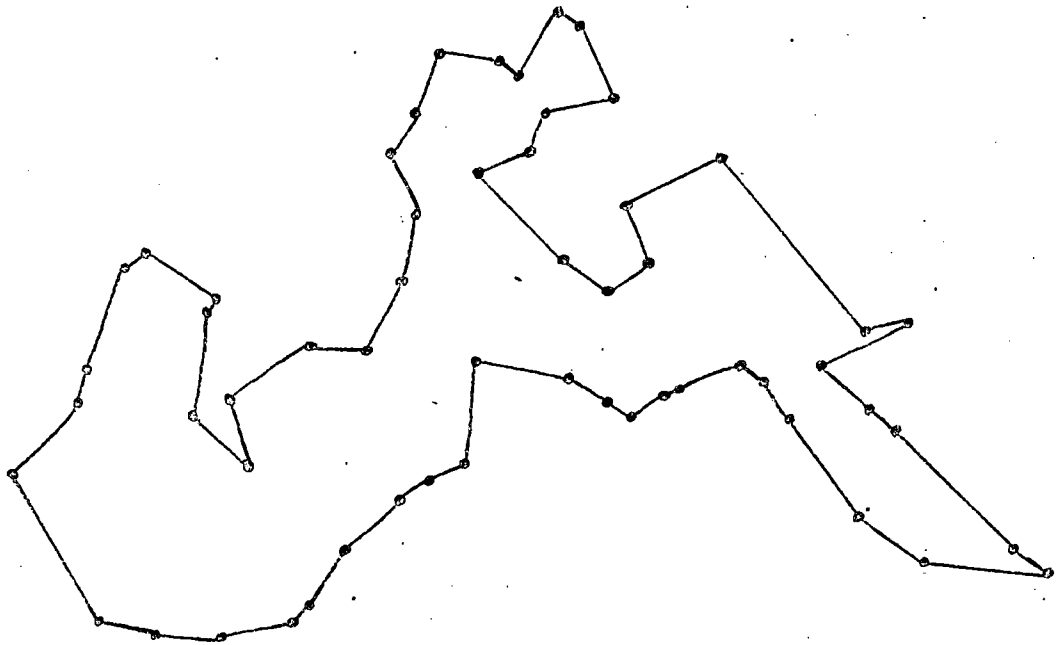


Figure 49: Synthesized Solution with Path Length 2021

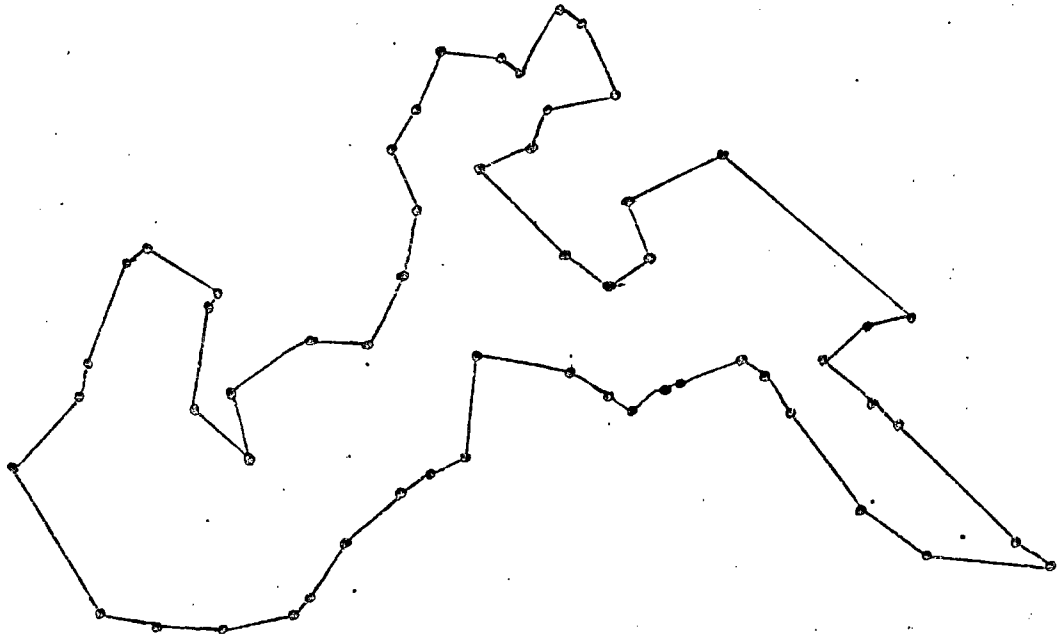


Figure 50: Synthesized Solution with Path Length 2012

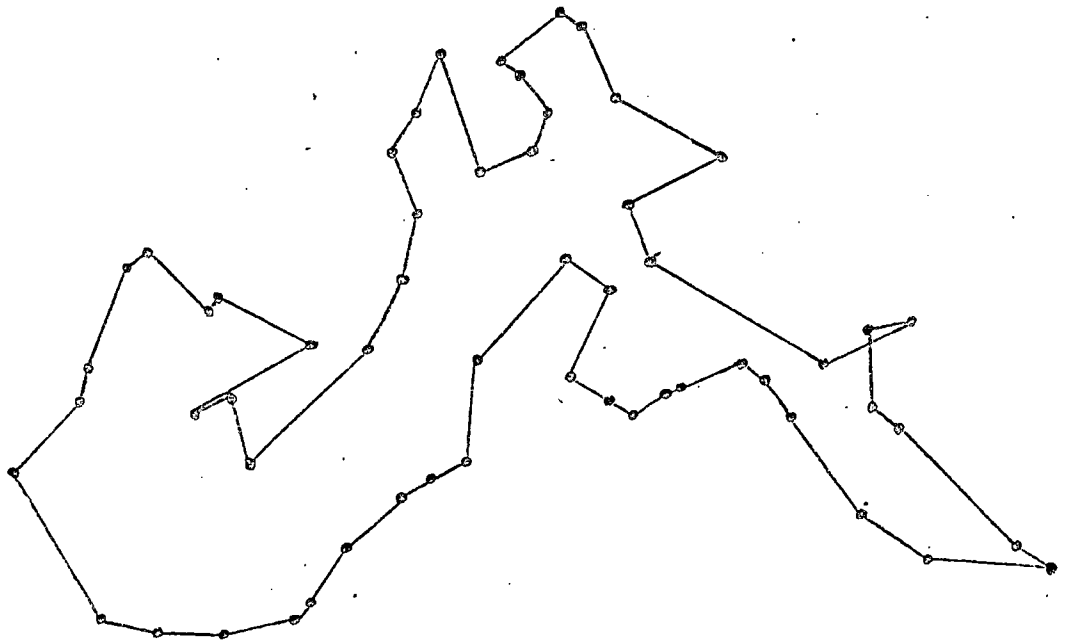


Figure 51: Croes Solution

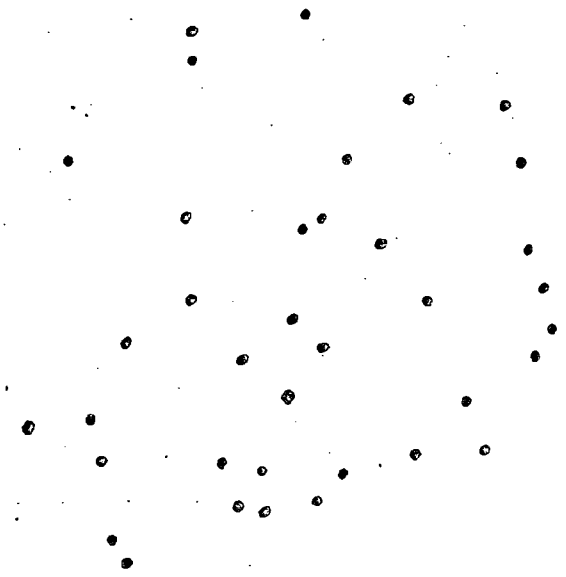


Figure 52: The Eire Travelling Salesman Problem

The process of communicating ideas in the system should be clear from the previous two examples. Only the subproblem decomposition solution, and the synthesized solution displays are reproduced for this problem.

Phase I The only general idea which occurred to us for this problem was to have an outer loop for the outside cities and an inner loop for the cities in the middle. To carry out the idea it was necessary to choose the outer and inner loop cities, and then, by choosing subproblem boundary points (endpoints) determine how the loops would be joined together.

Figure 53 displays the subproblem structure for what appeared to be the most obvious choice. The inter-tour joining links are not too long and the breaks at the joining points are relatively long indicating a good balance. This solution (figure 54) cost 13 CPU seconds to produce and 5 minutes sitting time. The path length is 1372.

Figure 55 is a display of the synthesized solution resulting from a different choice of "lower endpoint" for SP4. This change took 3 CPU seconds and 20 seconds elapsed time. The new path had a length of 1387.

We then decided to try increasing the contents of the inner loop and rejoining it to the reduced outer loop in roughly the same way (figure 56). The synthesized result is shown in figure 57. This is the same solution as in figure 55. With a little thought we could have predicted this. This carelessness cost 17 CPU seconds and 5 minutes elasped time.

Almost for want of something better to do we then tried several other ways of connecting the two loops. Figure 58 resulted from trying to join the two loops at the top (path length 1444, CPU usage 24 seconds, elapsed time 7 minutes). Joining the loops at the top turned out rather badly so

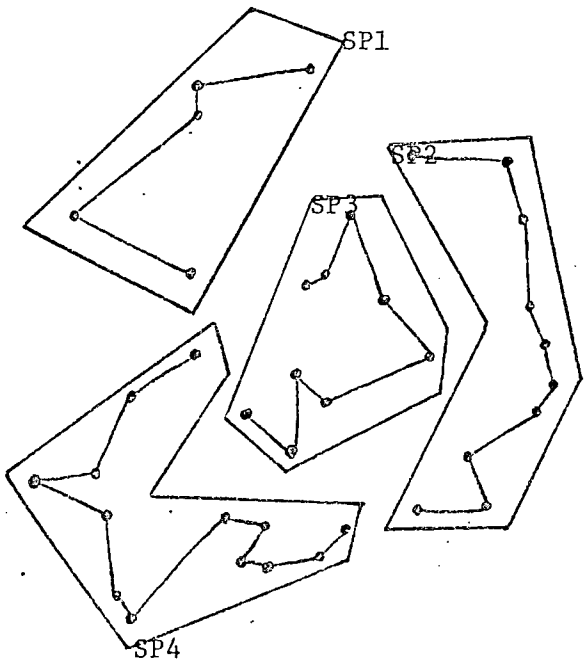


Figure 53: First Choice of Subproblems and
Their Solutions

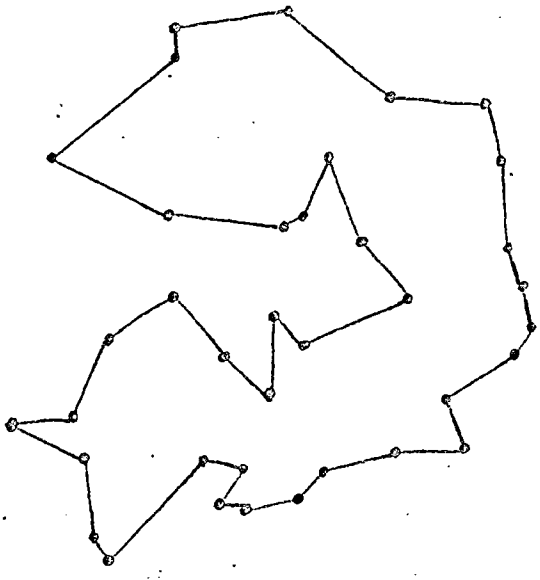


Figure 54: Synthesized Solution with
Path Length 1372

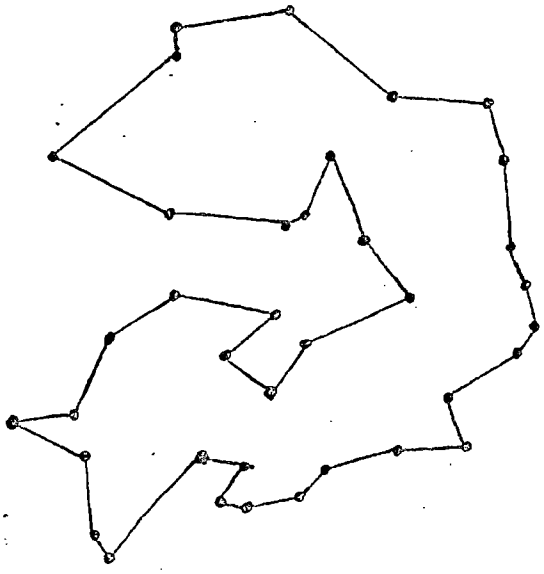


Figure 55: Synthesized Solution with Path Length 1387

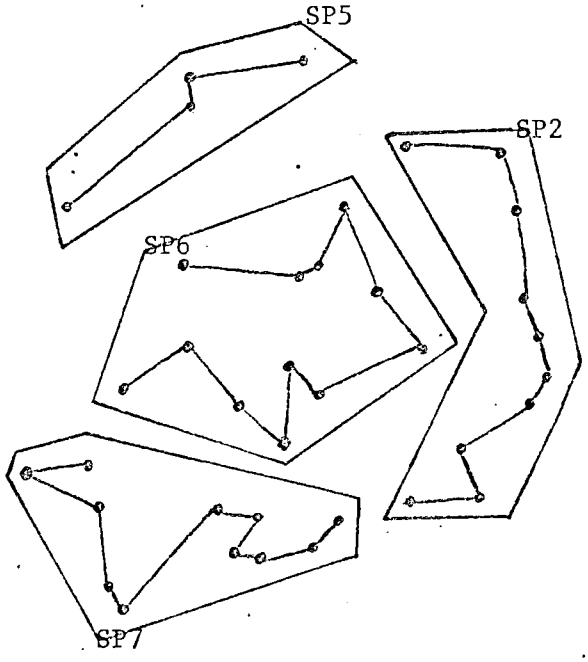


Figure 56: New Choice of Inner Loop Subproblem

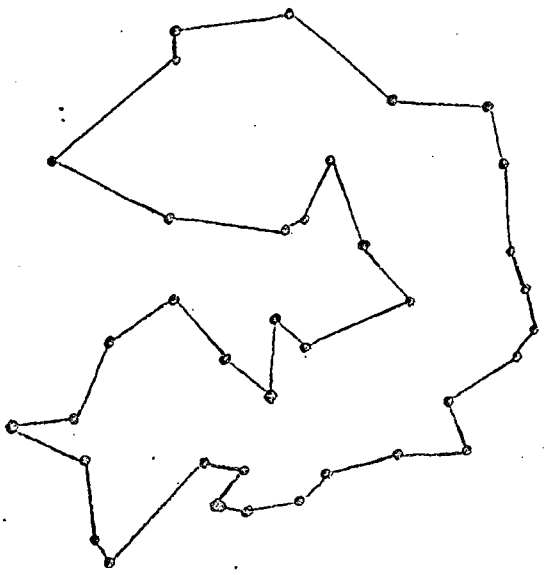


Figure 57: Synthesized Solution with Path Length 1387

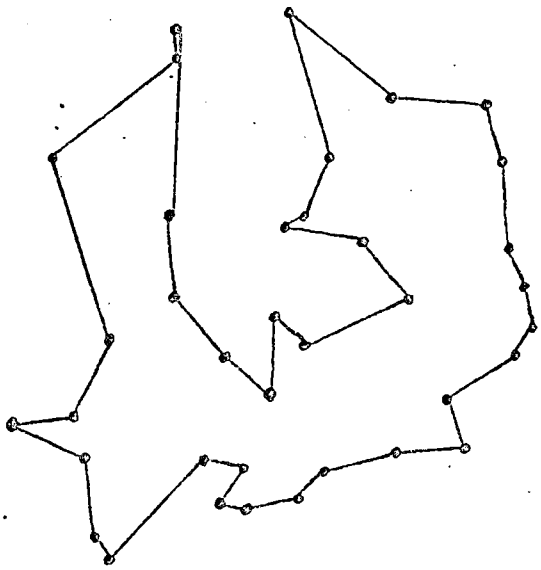


Figure 58: Synthesized Solution with Path Length 1444

we tried joining them at the bottom. The first choice (figure 59) resulted in a path length of 1403 and required 22 additional CPU seconds and an elapsed time of 7 minutes. The second choice (figure 60) had a path length of 1399 (15 CPU seconds and 4 minutes elapsed or sitting time).

Phase III The automatic, or single subproblem, solutions are displayed in figures 61 and 62. The first, produced by CCSOL has a path length of 1677 and required 54 CPU seconds to produce. The CLSOL solution was the same as our first interactive solution. It required 150 CPU seconds for its computation.

12. Conclusions

In this section we describe some of the conclusions we were able to come to on the basis of our experiments with the TSP system.

(a) Communication Medium. All of our experiments confirmed that the subproblem or planning approach is a natural, efficient way to structure the interactive process. The user is able to conveniently express and investigate general solution ideas. In the South American Problem he was able to first structure the solution as a "costal tour plus excursions"; in the France, Spain and Italy problem he could easily choose to build a solution around a natural grouping of the cities; and in the Eire problem he was able to investigate a general solution structure for a problem which was initially thought to be structureless. These examples were our first three experiments with the system and were not chosen from a set of larger, less fruitful or successful experiments.

The success of the planning approach for the TSP is based on the

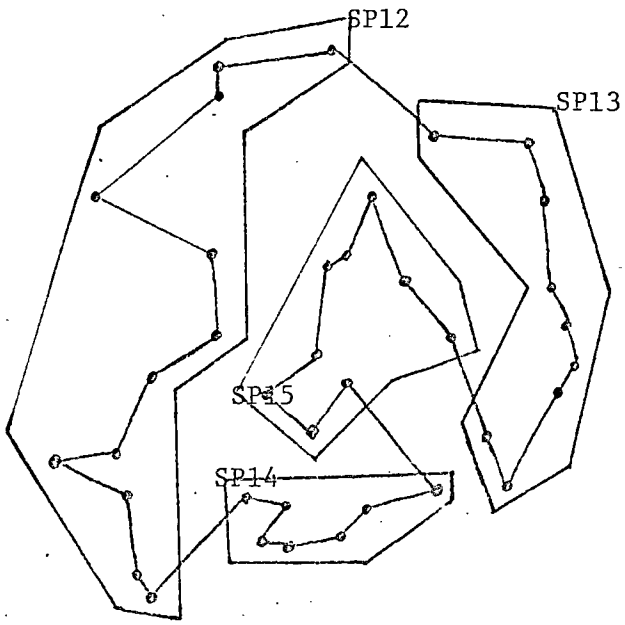


Figure 59: Synthesized Solution with Path Length 1403

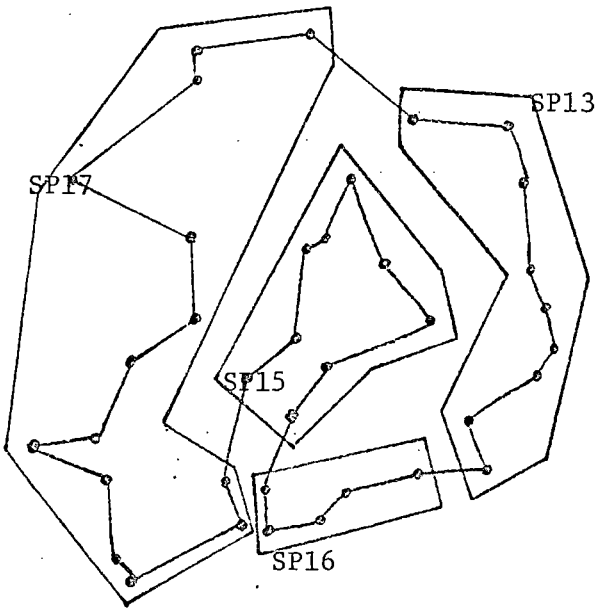


Figure 60: Synthesized Solution with Path Length 1399

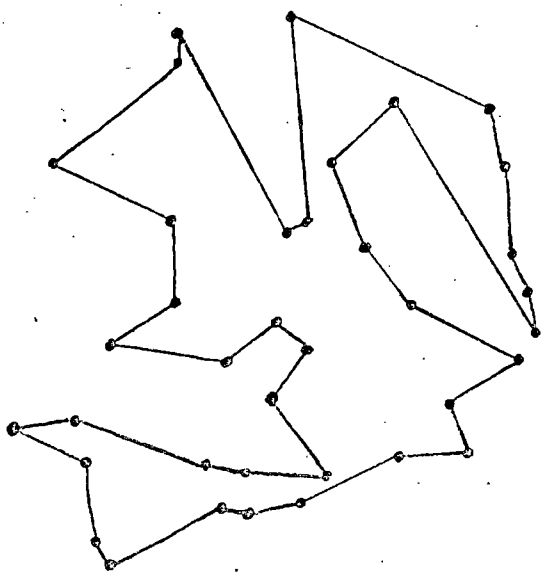


Figure 61: Croes Solution

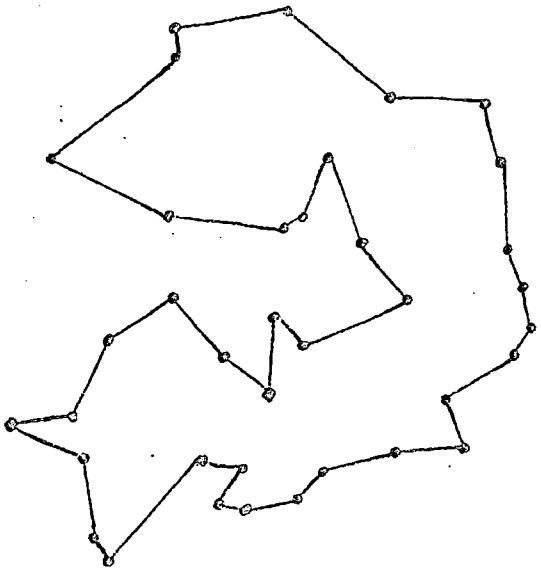


Figure 62: Lin Solution

ability of the user in the system to define "detail resistant" aspects of a solution with "large general" commands rather than "petty detailed" commands. The imposition of general solution ideas on the solution process occurs in two obvious ways. The first is through the definition of a subproblem. If the user recognizes a subproblem for which the solution is obvious (e.g. SP1 in figure 19) he can immediately isolate that part of the problem in a subproblem and apply the best suited solution process. In the TSP examples this took the form of applying a fast heuristic procedure to an obvious grouping of cities. In other cases the user will recognize subproblems without knowing exactly what their solutions should be (e.g. SP1 in figure 40). This confirms the claim that people have the type of general solution ideas which we have defined as R-Plan non-terminals. If the user were forced to express his ideas without the ability to define subproblems rather than solution pieces he would become hopelessly lost in the details of the alternative solutions to subproblems.

The second way in which the user expresses his general solution ideas is in terms of the subproblem or plan structure itself. In the South American problem, for example, the two abstract ideas for a solution, coastal-tour-and-excursions and two-circular-tours, were both communicated to the computer in terms of a subproblem structure.

We note that the user should not only be able to express general or high level solution ideas but that he be able to and encouraged to define properties of solutions which are robust against disturbance by detail. The two types of general solution properties described above have this property. Choices of subproblems and problem structures are based on patterns which admit many changes in detail. If the users role in a system is not one in which he can make, without penalty, small slips in detail, he would soon be lost either in an accumulation of errors or a sea of alternatives con-

structed in order to avoid error.

To better understand the importance of the detail resistant quality of a users ideas, consider the problem defined in figure 63 in which the cities are points which are "almost" on a grid.

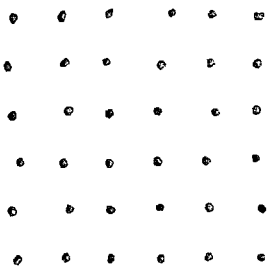


Figure 63: Cities almost on grid points.

The obvious idea for a solution is that illustrated in figure 64.

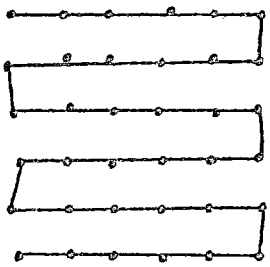


Figure 64: Obvious solution idea.

The optimal solution is shown in figure 65.

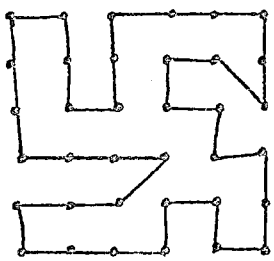


Figure 65: Optimal Solution.

The solution idea in this case is not resistant to disturbance by detail. It is based on a pattern consisting of a single problem. We do not mean to imply that the planning structure of the system will protect users from problems such as this. It is easy to define the idea for this solution in terms of subproblems even though it does not intuitively have true subproblem structure. We only mean to emphasize the importance of allowing the user to suggest solution ideas free from such detail and that subproblem and subproblem structure ideas often have this property.

(b) Psychological Advantages. One advantage of an interactive system with a sound communication structure is the psychological advantages it has over an automatic solver. By using such an interactive system the user can be assured that no obvious (to him) solution ideas have been overlooked. Heuristic programs are such that they may fail in an unpredictable and even undetectable way. The Lin TSP procedure, for example, will only check the optimality of a solution up to its "3-optness". It will not discover "obvious" 4-opt changes that could improve the solution. In the South American problem the user was able to construct a solution which was better than Lin's 3-opt solution for precisely this reason. In our interactive system the user is responsible for and hence can have an intimate knowledge of the solution structure.

If the system is used to "check out" automatically produced solutions it not only provides a psychological advantage but becomes part of a more powerful combinatorial tool. An additional facility in the system would be to allow the user to request that the computer "check out" an interactively produced solution. In constructing a solution interactively the user may make small errors in detail. The ability to "run" a user solution through a "hill climbing" procedure to check for the possibility of detailed improvements would provide an additional psychological

advantage for the system.

(c) Learning Environment. We found the interactive TSP system to provide a good problem solving learning environment. When we first began the construction of the system we had only one real idea of how to structure travelling salesman solutions. If a problem consisted of several concentrated groups of cities which were widely separated then subproblems could be created for each of the groups; we know that at least in this case the problem could be efficiently solved in a subproblem interactive system. Our experience with the South American Problem quickly revealed the importance of smooth circular tours. In all of the experiments the strategy of grouping cities in such tours proved effective. In addition, we gained confidence in the following rule of thumb for choosing inter-subproblem endpoint pairs: "the closest pair of endpoints is usually the optimal choice for inter-subproblem linking. We predict that further experience with the system will further increase a users knowledge of the TSP and hence enhance the power of the system as a problem solving tool. There is no indication that this process of solver growth through learning will ever be possible in a completely automatic solution process.

(d) Combinatorial Tool. We feel that we have proved the validity of planning method as an approach to several important problems in man-machine communication. The power of the system as a problem solving tool must now be considered. Is it worth the bother to allow man machine communication? Certainly the interactive system allows the user to spend his combinatorial power where he chooses, and to construct solutions which, although sub-optimal, are still meaningful to him. But does it produce better solutions? This question can be answered with reference both to the quality of the solutions produced and to the cost of producing solutions.

Our experience indicates that the system provides good quality solutions. In each of the three experiments described in the report the interactive solution was at least as optimal as that produced by an automatic problem solver. In the South American problem the interactive solution was marginally better (2733 vs 2741). In general, the indications are that for problems in the range of 30-70 cities the interactive solutions will be good but no better than a solution produced by the Lin procedure.

With respect to solution quality, the interactive system is probably best considered as a method for extending the range of presently available solvers to larger problems. The relatively poor solutions produced by the Croes procedure indicate the interactive approach to be a better strategy to adopt than to use less discriminating automatic procedures for problems too large to be attacked with powerful, but expensive, automatic procedures.

By default the interactive solver is always capable of producing at least as good a solution as any automatic problem solver. It is only necessary to incorporate the automatic procedure as a subproblem solver and then apply it to the subproblem consisting of the whole problem.

The real advantage of the interactive system as a combinatorial tool can be measured in terms of the cost of solution production. Experience indicates that for large problems with subproblem structure interactive solutions are significantly cheaper to produce. For very large problems the planning interactive approach may be the only method of constructing reliably optimal solutions.

Solutions are produced less expensively in the system by trading off the cost of constructing subproblems against the displaced cost of solving whole or undivided problems. Consider the figures for the South American

problem in figure 66. We arrived at the solution cost for the interactive solutions by adding the accumulated CPU usage and a factor for elapsed or "sitting time". For no particularly good reason we equated the cost of one hour elapsed time with 100 CPU seconds. The relative differences between the figures are large enough to allow a quite different equation without invalidating the conclusions. We note that wide variances in the CPU usage reported by the accounting system can result from varying loads on the machine. We therefore took care to perform the experiments under the same system loading conditions.

	Path Length	Solution Cost
First Interactive	2811	244
Final Interactive	2733	521
Croes' Solution	3219	540
Lin's Solution	2741	1296

Figure 66: Performance figures for the South American TSP

From the figures in 66 we see that Lin's solution is 2.5% better than the first interactive solution but at a 431% increase in the cost of solution production. In this example the final interactive solution was better than Lin's solution and 200% cheaper to produce.

The dramatic relative cost efficiency of the interactive system for the South American TSP results from the replacement of a very large problem (68 cities) with several smaller subproblems. Recall that the Lin solution of a TSP increases in computation time wth the cube of the increase in the number of cities.

The figures for the France, Spain and Italy problem are displayed in figure 67.

	Path Length	Solution Cost
First Interactive	2094	65
Final Interactive	2012	328
Croes' Solution	2140	169
Lin's Solution	2012	814

Figure 67: Performance figures for the France, Spain and Italy TSP

The Lin solution is 4% better than the first interactive solution at a 1200% increase in solution cost. The final interactive solution was the same as Lin's solution and 188% cheaper to construct.

The figures for the Eire problem are shown in figure 68. The figures do not at first appear quite as encouraging. Although the first interactive solution was as good as the Lin solution, and at an 85% decrease in solution cost, the final interactive solution, again the same as the Lin solution, was 7% more expensive to produce. We note that we could have reduced the interaction cost by creating smaller subproblems than were really required to express the interactive solution ideas. This is an important added advantage of the interactive approach: the user can control not only the structure of the solution produced but also the cost of solution production. By constructing subproblems of different sizes he can trade off elapsed time plus cost of subproblem construction against cost of subproblem solution computation.

	Path Length	Solution Cost
First Interactive	1372	22
Final Interactive	1372	160
Croe's Solution	1576	55
Lin's Solution	1372	150

In comparing the Lin solution computation cost to the interactive solution cost there are grounds for an argument that the comparison should be made with the first interactive solution. In addition, since the Lin procedure is heuristic, and produces different solutions for different runs, the final interactive solution should be compared with a final Lin solution. A final Lin solution would be chosen from a number of executions of the Lin procedure. This can be accomplished either by increasing the number of 3-opt cycles per reduction cycle or by just calling the procedure several times with the default setting of two 3-opts per reduction cycle. We decided on the second alternative and computed three additional Lin solutions. The path lengths were 1390 (figure 69), 1390 and 1372. When the Lin procedure constructed the original solution (path length 1372) we had the same degree of confidence in the solution as we had for our final interactive solution. The costs of producing these additional solutions were 114, 146, and 184 CPU seconds. The cost of the final Lin solution is therefore 594. If we compare the final Lin solution with the final interactive solution then the same reduced costs for this problem as for the other two can be claimed. It is unlikely that we could have even afforded to compute a final Lin solution for the South American problem.

(e) System Limitations and Possible Extensions. In this section some of the limitations mentioned in the three experiments are reviewed.

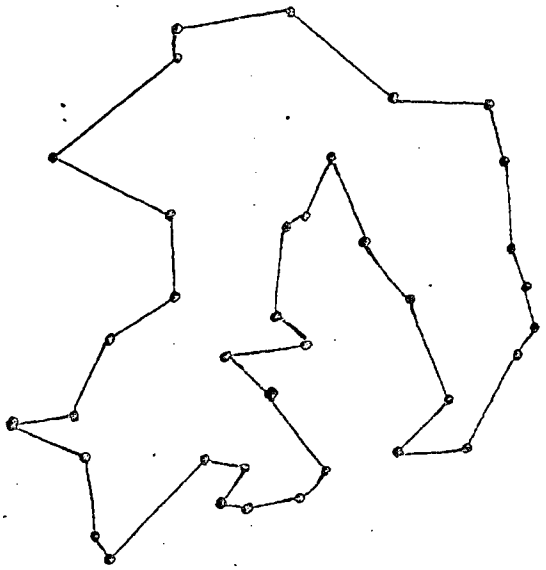


Figure 69: Alternative Lin Solution to Eire Problem

The first obvious deficiency was the lack of any convenient method for "manually" constructing subproblem solutions. In several instances subproblem solvers failed to produce the desired results for high level subproblems. The required addition to the present system would be a new subproblem solver, say CUSOL, which allowed the user to define the exact ordering of a subproblem solution.

A more general limitation of the system is the result of what we have called the context problem. In the present system the subproblem solvers find solutions which are optimal for a given choice of endpoints. It is necessary, however, to find subsolutions which are optimal with respect to their path lengths plus the cost of linking the solution to neighbouring solutions. In other words, subsolutions must be optimal with respect to the context in which they occur. The automatic parts of the present system (i.e. the subproblem solvers) and not sensitive to subproblem context in any way: the user is entirely responsible for determining the choice of endpoints that optimizes the subproblem solution while still allowing inexpensive interproblem linking. Very often this decision results from the attempted comparison of combinations of single links. The user can either attempt to perform these comparisons visually, or calculate individual distances and then perform the necessary arithmetic combinations using pencil and paper. Neither alternative is satisfactory.

Several possibilities can be proposed to eliminate the context problem. Two suggestions were described in Phase I of the solution of the South American Travelling Salesman Problem. Another suggestion is to change the system so that subproblems are not disjoint. If each subproblem in a tour of subproblems had one city in common with the next subproblem, there would be no such thing as an inter-subproblem link. There are a number of disadvantages to this idea. It implies that the user must decide at the time of subproblem creation how the subproblems are to be joined together.

He cannot delay the decision and choose interproblem links at some later point. Subproblems would all be one order of magnitude larger than in the present system (they would all have one more city). It is not clear that a system based on overlapping subproblems would not introduce a whole set of new limitations or even introduce a more severe context problem. It would certainly provide a much less flexible system. The ability to manipulate subproblem endpoints independently of context would be lost since each subproblem solution endpoint would belong to two subproblems. The importance of being able to manipulate individual subsolution properties through the subproblem endpoints is illustrated in the solution process of the Phase III part of the solution to the France, Spain and Italy problem.

Perhaps the best solution to the context problem would be to construct subproblem solvers which accepted as an argument a list of alternative subproblem endpoints and alternative neighbouring subproblem endpoints. The solvers would return the solution which was optimal with respect to the subsolution path length between solution endpoints plus the lengths of two links to endpoints in neighbouring subproblems. The solver would also return the names of the optimal endpoints in the neighbouring subproblems. If the neighbouring subproblems were already solved, so that the choice of neighbour endpoints had already been fixed, then the users alternatives would be limited to different choices of solution endpoints for the subproblem to be solved. In this approach the user maintains control over solution structure through his responsibility for naming alternative endpoint choices. The approach could be implemented in the present system with the definition of several new commands, all of which could be defined in terms of presently available commands. Other,

higher level, commands could be defined in terms of these commands and the system would still retain its flexible approach to endpoint manipulation.

A second general limitation in the system is the lack of any facility for dealing with alternative solution decisions. At present the user must keep track of alternatives. The second suggested solution to the context problem described above would also solve part of the alternatives problem. It would permit the user to request a solver to return the best of a number of alternative subproblem solutions defined by a number of alternative endpoint choices.

Occasionally a user will want to return to an earlier point in the solution process and take up some other unexplored alternative. Suppose, for example, there had been two possible choices for a subproblem group of cities at that point (cf. Phase I of the South American problem). At present the user must remember what the alternative was, "undo" the present solution, and reconstruct the previous partial solution situation. What is required in this case is a facility for checkpointing partial solutions and a partial solution classification and retrieval scheme.

(f) Summary. In conclusion, we found the planning approach to provide a sound basis for an interactive system. In a planning system the user can communicate general or abstract ideas to the computer. He can control the cost of solution production in a semantically meaningful way through subproblem definition. In the case of the TSP he is able to produce good quality solutions and there is every indication that the approach allows the design of a good problem solving learning environment.

The limitations we discovered during our experience with the system could all be eliminated while maintaining the same planning or subproblem

basis for the system. Even in its present form, however, the system was sufficient for the purposes of our investigation. These limitations are described to assist either in the construction of a production form of this system or in the construction of systems for other problems.

We note the importance of reliable equipment for interactive systems. The accounts in sections 9, 10 and 11 are those of experiments where hardly anything went wrong. The action required when a malfunction occurred has not been included in the account. For the most part this consisted of repeating a command which went astray.

Acknowledgements The author wishes to express his gratitude to Professor Julian Feldman for his continuing encouragement and enlightening criticism.

Appendix I City coordinate pairs for the three experiments.

```
(DEFPROP CITIES
 (CITIES (584. 624.)
         (616. 816.)
         (643. 804.)
         (487. 659.)
         (567. 732.)
         (519. 664.)
         (684. 743.)
         (728. 728.)
         (783. 711.)
         (815. 680.)
         (816. 671.)
         (815. 660.)
         (807. 647.)
         (776. 616.)
         (599. 604.)
         (679. 599.)
         (716. 568.)
         (716. 551.)
         (747. 543.)
         (723. 523.)
         (695. 511.)
         (679. 512.)
         (464. 592.)
         (495. 571.)
         (492. 560.)
         (448. 555.)
         (451. 515.)
         (495. 515.)
         (568. 507.)
         (648. 479.)
         (624. 457.)
         (612. 435.)
         (451. 479.)
         (559. 484.)
         (568. 452.)
         (576. 415.)
         (560. 412.)
         (543. 431.)
         (475. 428.)
         (508. 443.)
         (448. 455.)
         (451. 427.)
         (456. 419.)
         (443. 384.)
         (527. 371.)
         (520. 351.)
         (448. 353.)
         (448. 336.)
         (496. 303.)
         (488. 240.)
         (484. 224.)
         (482. 855.)
         (448. 856.)
         (441. 839.)
         (407. 800.)
         (399. 851.)
         (391. 808.)
```

```
(351. 719.)
(364. 747.)
(416. 716.)
(352. 679.)
(364. 663.)
(391. 623.)
(395. 627.)
(436. 620.)
(440. 604.)
(448. 592.)
(448. 583.))
VALUE)
```

```
(DEFPROP CITIES
 (CITIES (375. 304.)
         (352. 295.)
         (383. 268.)
         (415. 332.)
         (364. 347.)
         (368. 354.)
         (332. 376.)
         (320. 367.)
         (304. 323.)
         (296. 300.)
         (264. 264.)
         (307. 191.)
         (336. 184.)
         (375. 183.)
         (407. 188.)
         (415. 199.)
         (432. 224.)
         (463. 256.)
         (476. 264.)
         (495. 271.)
         (444. 328.)
         (460. 363.)
         (455. 428.)
         (467. 395.)
         (499. 415.)
         (527. 428.)
         (468. 448.)
         (480. 480.)
         (511. 476.)
         (532. 448.)
         (519. 467.)
         (539. 500.)
         (552. 492.)
         (568. 456.)
         (623. 424.)
         (575. 404.)
         (536. 368.)
         (495. 320.)
         (543. 316.)
         (564. 304.)
         (575. 296.)
         (596. 307.)
         (564. 360.)
         (588. 375.)
         (599. 308.)
         (631. 323.)
         (728. 344.)
         (695. 340.)
         (672. 323.)
         (643. 316.)
         (656. 295.)
         (696. 299.)
         (706. 291.)
         (767. 251.)
         (791. 216.)
         (691. 248.)
         (727. 224.))
```

```
(DEFPROP CITIES
 (CITIES (495. 472.)
         (472. 544.)
         (524. 500.)
         (575. 499.)
         (582. 468.)
         (512. 428.)
         (536. 400.)
         (591. 372.)
         (600. 387.)
         (595. 408.)
         (587. 428.)
         (416. 520.)
         (415. 535.)
         (351. 467.)
         (415. 439.)
         (471. 436.)
         (484. 443.)
         (415. 400.)
         (467. 391.)
         (484. 376.)
         (556. 352.)
         (567. 327.)
         (531. 324.)
         (467. 351.)
         (440. 371.)
         (383. 379.)
         (367. 340.)
         (332. 335.)
         (371. 319.)
         (428. 319.)
         (452. 316.)
         (495. 315.)
         (480. 300.)
         (455. 295.)
         (440. 296.)
         (376. 279.)
         (387. 268.))
VALUE)
```

Appendix II Frame numbers for display information.

Frame	Displayed Information
1	City points
2	Subproblem Polygons
3	Miscellaneous
4	Context Marker (C)
5	Subproblem Solution Links
6	Synthesized Solution Links
7	Pen pointer marker (P)

Appendix III The Karp Solver CKSOL

```
(DEFPROP LKARP
 (LKARP ILIST1
        COSTK
        DIS1
        CKSOL
        IEXP
        SOLVER1
        SQRT
        MIN2
        MIN1
        DELETE
        ELEMENT
        ILIST
        INDEX
        LKARP
        FAC
        FACSEG
        SETBINOM
        SQ
        MIN3)
VALUE)

(DEFPROP ILIST1
 (LAMBDA(A B)
  (COND ((*LESS B A) NIL) (T (CONS B (ILIST1 A (SUB1 B))))))
EXPR)

(DEFPROP COSTK
 (LAMBDA(L)
  (PROG (COUNT)
        (COND ((NULL L) (RETURN NIL)))
        (SETQ COUNT 0)
   LAB  (COND ((NULL (CDR L)) (RETURN COUNT)))
        (SETQ COUNT (*PLUS COUNT (CDISA (CAR L) (CADR L))))
        (SETQ L (CDR L))
        (GO LAB)))
EXPR)

(DEFPROP DIS1
 (LAMBDA(CC1 CC2)
  (SQRT
   (*PLUS (SQ
            (*DIF (CAR (GET CC1 (QUOTE CENTROIDP)))
                  (CAR (GET CC2 (QUOTE CENTROIDP)))))
          (SQ
           (*DIF (CADR (GET CC1 (QUOTE CENTROIDP)))
                 (CADR (GET CC2 (QUOTE CENTROIDP))))))))
EXPR)

(DEFPROP CKSOL
 (LAMBDA(NAME)
  (PROG (CITIES L IN OUT COST)
        (SETQ NAME (CAR (POINTER NAME)))
        (SETQ CITIES (GET NAME (QUOTE OBJECTSP)))
        (COND ((NULL CITIES) (RETURN NIL)))
        (COND ((EQ NAME (QUOTE UNIV))
               (SETQ IN (CAR CITIES))
               (SETQ OUT IN)
```

```
                (GO LAB))
               (T (CDPTS1 NAME)))
         (SETQ IN (CAAR (GET NAME (QUOTE STRUCTUREP))))
         (SETQ OUT (CADAR (GET NAME (QUOTE STRUCTUREP))))
         (SETQ CITIES (CONS IN (REMOVEX IN CITIES)))
   LAB   (COND ((EQ IN OUT)
                (SETQ CITIES (APPEND CITIES (LIST OUT))))
               (T
                (SETQ CITIES
                      (APPEND (REMOVEX OUT CITIES)
                              (LIST OUT)))))
         (SETQ NUM (LENGTH CITIES))
         (COND ((*LESS NUM 4) (GO LAB1))
               ((*GREAT NUM MAXKAPP)
                (RETURN
                 (QUOTE (MAXIMUM NUM OF CITIES EXCEEDED)))))
         (SETQ CITIES (SOLVER1 CITIES))
   LAB1  (PUTPROP NAME
                  (LIST (LIST IN OUT) CITIES)
                  (QUOTE STRUCTUREP))
         (DSOL NAME)
         (RETURN COST)))
EXPR)

(DEFPROP IEXP
 (LAMBDA(I J)
  (COND ((ZEROP J) 1) (T (*TIMES I (IEXP I (SUB1 J)))))))
EXPR)

(DEFPROP SOLVER1
 (LAMBDA(CITIES)
  (PROG (D I X C L LEV CITYNAMES J)
        (SETQ X (TIMES (*DIF NUM 2) (IEXP 2 (*DIF NUM 3))))
        (SETQ I 0)
   LAB3 (STORE (TABLE I) 0)
        (COND ((*LESS I X) (SETQ I (ADD1 I)) (GO LAB3)))
        (SETQ CITYNAMES CITIES)
        (SETQ I 0)
        (MAP
         (FUNCTION
          (LAMBDA(X)
           (PROG NIL
                 (SETQ I (ADD1 I))
                 (STORE (CLIST I) (CAR X))
                 (COND
                  ((NULL (CDR X)) (RETURN NIL))
                  (T (SETQ J I)
                     (MAP
                      (FUNCTION
                       (LAMBDA(Y)
                        (PROG (D)
                              (COND
                               ((NULL Y) (RETURN NIL))
                               (T (SETQ J (ADD1 J))
                                  (COND
                                   ((NILL
                                     (AND
                                      (GET
                                       (CAR Y)
                                       (QUOTE TERMINALP))
```

```
                                        (GET
                                         (CAR Y)
                                         (QUOTE TERMINALP))))
                                    (SETQ
                                     D
                                     (DIS1 (CAR X)
                                           (CAR Y))))
                                   (T (SETQ CC1
                                            (EVAL (CAR X)))
                                      (SETQ CC2
                                            (EVAL (CAR Y)))
                                      (COND
                                       ((*LESS CC1 CC2)
                                        (SETQ
                                         D
                                         (DST CC1 CC2)))
                                       (T
                                        (SETQ
                                         D
                                         (DST CC2 CC1))))))
                                (STORE (CDISA I J) D)
                                (STORE (CDISA J I)
                                       D
                                       (RETURN NIL)))))))
                        (CDR X)))
                  (RETURN NIL))))
          CITIES)
         (SETQ L (ILIST 2 NUM))
         (SETQ LEV (SUB1 NUM))
         (MIN2 LEV L LEV)
         (SETQ L (MIN3))
         (SETQ COST (COSTK L))
         (SETQ CITIES NIL)
    LAB2 (SETQ X (CAR L))
         (SETQ L (CDR L))
         (SETQ CITIES (CONS (CLIST X) CITIES))
         (COND ((NULL L) (RETURN (REVERSE CITIES))))
         (GO LAB2)))
 EXPR)

(DEFPROP SORT
 (LAMBDA(X)
  (PROG (V1 V2)
        (SETQ X (*PLUS X 0.0))
        (COND ((ZEROP X) (RETURN 0)))
        (SETQ V1 (*QUO X 2))
        (SETQ V2 (*QUO (*PLUS V1 (*QUO X V1)) 2))
    LAB (COND
         ((*LESS (ABS (*DIF V2 V1)) 1.0E-2)
          (RETURN (FIX V2))))
        (SETQ V1 V2)
        (SETQ V2 (*QUO (*PLUS V1 (*QUO X V1)) 2))
        (GO LAB)))
 EXPR)

(DEFPROP MIN2
 (LAMBDA(LEV L P)
  (PROG (M X END P1 I)
        (SETQ LEV (SUB1 LEV))
        (SETQ M 63000)
```

```
        (SETQ X (DELETE P L))
        (SETQ END (CAR X))
        (SETQ L (CADR X))
        (SETQ P1 Ø)
        (SETQ I (INDEX LEV L))
   LAB  (COND ((EQ P1 LEV) (RETURN M)))
        (SETQ P1 (ADD1 P1))
        (SETQ X
              (*PLUS (MIN1 LEV L P1 I)
                     (CDISA (ELEMENT P1 L) END)))
        (COND ((*LESS X M) (SETQ M X)))
        (GO LAB)))
EXPR)

(DEFPROP MIN1
 (LAMBDA(LEV L P I)
  (PROG (X Y)
        (SETQ I (*PLUS I P))
        (SETQ Y (TABLE I))
        (COND
         ((ZEROP Y)
          (COND
           ((EQ LEV 2)
            (COND
             ((EQ P 1)
              (SETQ
               X
               (*PLUS (CDISA 1 (CADR L))
                      (CDISA (CADR L) (CAR L)))))
             (T
              (SETQ
               X
               (*PLUS (CDISA 1 (CAR L))
                      (CDISA (CAR L) (CADR L)))))))
           (T (SETQ X (MIN2 LEV L P))))
          (STORE (TABLE I) X)
          (RETURN X)))
        (RETURN Y)))
EXPR)

(DEFPROP DELETE
 (LAMBDA(P L)
  (PROG (I L1)
        (SETQ I 1)
        (SETQ L1 NIL)
   LAB  (COND
         ((EQ I P)
          (RETURN
           (LIST (CAR L) (NCONC (REVERSEX L1) (CDR L))))))
        (SETQ L1 (CONS (CAR L) L1))
        (SETQ L (CDR L))
        (SETQ I (ADD1 I))
        (GO LAB)))
EXPR)

(DEFPROP ELEMENT
 (LAMBDA(P L)
  (PROG (I)
        (SETQ I 1)
   LAB  (COND ((EQ I P) (RETURN (CAR L))))
```

```
        (SETQ L (CDR L))
        (SETQ I (ADD1 I))
        (GO LAB)))
EXPR)

(DEFPROP ILIST
 (LAMBDA (A B) (REVERSE (ILIST1 A B)))
EXPR)

(DEFPROP INDEX
 (LAMBDA(LEV L)
  (PROG (TOTAL1 TOTAL2 I X K Y HIGH)
        (SETQ X (*DIF NUM 2))
        (SETQ TOTAL1 0)
        (SETQ I 2)
   LAB1 (COND
         ((*LESS I LEV)
          (SETQ TOTAL1
                (*PLUS TOTAL1 (*TIMES I (BINOM X I))))
          (SETQ I (ADD1 I))
          (GO LAB1)))
        (SETQ TOTAL2 0)
        (SETQ K 1)
        (SETQ I 2)
        (SETQ HIGH (CAR L))
        (SETQ Y (SUB1 NUM))
   LAB  (SETQ X (*DIF LEV K))
   LAB2 (COND
         ((*LESS I HIGH)
          (SETQ TOTAL2 (*PLUS TOTAL2 (BINOM (*DIF Y I) X)))
          (SETQ I (ADD1 I))
          (GO LAB2)))
        (COND
         ((EQ K LEV)
          (RETURN (*PLUS (*TIMES TOTAL2 LEV) TOTAL1))))
        (SETQ L (CDR L))
        (SETQ K (ADD1 K))
        (SETQ I (ADD1 HIGH))
        (SETQ HIGH (CAR L))
        (GO LAB)))
EXPR)

(DEFPROP LKARP
 (LKARP ILIST1
        CUSTK
        DIS1
        CKSOL
        IEXP
        SOLVER1
        SQRT
        MIN2
        MIN1
        DELETE
        ELEMENT
        ILIST
        INDEX
        LKARP
        FAC
        FACSEQ
        SETBINOM
```

```
        SQ
        MIN3)
VALUE)

(DEFPROP FAC
 (LAMBDA(N)
  (COND ((ZEROP N) 1) (T (*TIMES N (FAC (SUB1 N)))))))
EXPR)

(DEFPROP FACSEG
 (LAMBDA(I J)
  (COND ((*LESS I J) (QUOTE FACSEG-UNDEFINED))
        ((EQ I J) 1)
        (T (*TIMES I (FACSEG (SUB1 I) J)))))
EXPR)

(DEFPROP SETBINOM
 (LAMBDA NIL
  (PROG (I J)
        (ARRAY BINOM T (ADD1 MAXKARP) (ADD1 MAXKARP))
        (SETQ I 0)
   LAB1 (COND ((*GREAT I MAXKARP) (RETURN NIL)))
        (SETQ J 0)
   LAB2 (STORE (BINOM I J)
               (*QUO (FACSEG I J) (FAC (*DIF I J))))
        (COND ((EQ J I) (SETQ I (ADD1 I)) (GO LAB1)))
        (SETQ J (ADD1 J))
        (GO LAB2)))
EXPR)

(DEFPROP SQ
 (LAMBDA (X) (*TIMES X X))
EXPR)

(DEFPROP MIN3
 (LAMBDA NIL
  (PROG (MINL MINP LEV L X MIN P)
        (SETQ MINL NIL)
        (SETQ MINP (SUB1 NUM))
        (SETQ LEV MINP)
        (SETQ L (ILIST 2 NUM))
   LAB1 (SETQ X (DELETE MINP L))
        (SETQ L (CADR X))
        (SETQ END (CAR X))
        (SETQ MINL (CONS END MINL))
        (SETQ LEV (SUB1 LEV))
        (COND
         ((EQ LEV 1) (RETURN (CONS 1 (CONS (CAR L) MINL)))))
        (SETQ MINP 1)
        (SETQ MIN 630000)
        (SETQ P 1)
   LAB2 (SETQ X (TABLE (*PLUS (INDEX LEV L) P)))
        (SETQ X (*PLUS X (CDISA (ELEMENT P L) END)))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ MINP P)))
        (COND ((EQ P LEV) (GO LAB1)))
        (SETQ P (ADD1 P))
        (GO LAB2)))
EXPR)
```

Appendix IV The Lin and Croes Solvers CLSOL and CCSOL

```
(DEFPROP LIN
 (LIN NMEMX
      NTHREEOPT
      CLSOL
      COSTL
      LIN1
      LIN2
      ECOPY
      ATHREEOPT
      REVERSEXX
      THREEOPT
      RANDOMI
      REVERSEX
      RPERM
      RND
      LIN)
VALUE)

(DEFPROP NMEMX
 (LAMBDA(X Y L)
  (PROG NIL
   LAB  (COND ((NULL L) (RETURN T))
              (T
               (COND ((OR (AND (EQ X (CAAR L))
                               (EQ Y (CADAR L)))
                          (AND (EQ Y (CAAR L))
                               (EQ X (CADAR L))))
                      (RETURN NIL))
                     (T (SETQ L (CDR L)) (GO LAB))))))))
EXPR)

(DEFPROP NTHREEOPT
 (LAMBDA(L)
  (PROG (L1 L2
            L3
            CC1
            CC2
            CC3
            CC4
            CC5
            CC6
            D13
            D12
            D14
            D15
            D24
            D25
            D26
            D34
            D35
            D36
            D46
            D56
            DD
            CL1
            CL2
            CL3
            CL4
```

```
          CL5
          CL6)
       (COMMENT THIS
                IS
                A
                NEW
                THREEOPT
                ALGORITHM
                COMPUTES
                THREEOPTS)
       (COMMENT BY RANDOM IMPROV NOT STEEPEST ASCENT)
LAB0   (SETQ L1 L)
LAB1   (COND ((NULL (CDR L1)) (RETURN L)))
       (SETQ CC1 (CAR L1))
       (SETQ CL1 L1)
       (SETQ CL2 (CDR CL1))
       (SETQ CC2 (CAR CL2))
       (SETQ D12 (CDISA CC1 CC2))
       (COND ((MINUSP D12) (SETQ L1 (CDR L1)) (GO LAB1)))
       (SETQ L2 (CDR L1))
LAB2   (COND
        ((NULL (CDR L2)) (SETQ L1 (CDR L1)) (GO LAB1)))
       (SETQ CC3 (CAR L2))
       (SETQ CL3.L2)
       (SETQ CL4 (CDR CL3))
       (SETQ CC4 (CAR CL4))
       (SETQ D34 (CDISA CC3 CC4))
       (COND ((MINUSP D34) (SETQ L2 (CDR L2)) (GO LAB2)))
       (SETQ D13 (CDISA CC1 CC3))
       (SETQ D14 (CDISA CC1 CC4))
       (SETQ D24 (CDISA CC2 CC4))
       (SETQ L3 (CDR L2))
LAB3   (COND
        ((NULL (CDR L3)) (SETQ L2 (CDR L2)) (GO LAB2)))
       (SETQ CC5 (CAR L3))
       (SETQ CL5 L3)
       (SETQ CL6 (CDR CL5))
       (SETQ CC6 (CAR CL6))
       (SETQ D56 (CDISA CC5 CC6))
       (COND ((MINUSP D56) (SETQ L3 (CDR L3)) (GO LAB3)))
       (SETQ DD (*PLUS D12 (*PLUS D34 D56)))
       (COND
        ((*GREAT DD (*PLUS D13 (*PLUS D24 D56)))
         (RPLACD CL1 CL3)
         (REVERSEXX CL2 CL3)
         (RPLACD CL2 CL4)
         (GO LAB0)))
       (SETQ D26 (CDISA CC2 CC6))
       (SETQ D35 (CDISA CC3 CC5))
       (COND
        ((*GREAT DD (*PLUS D14 (*PLUS D26 D35)))
         (RPLACD CL1 CL4)
         (REVERSEXX CL2 CL3)
         (RPLACD CL2 CL5)
         (RPLACD CL5 CL3)
         (GO LAB0)))
       (SETQ D15 (CDISA CC1 CC5))
       (SETQ D36 (CDISA CC3 CC6))
       (COND
        ((*GREAT DD (*PLUS D24 (*PLUS D36 D15)))
```

```
          (RPLACD CL1 CL5)
          (REVERSEXX CL4 CL5)
          (RPLACD CL4 CL2)
          (RPLACD CL3 CL6)
          (GO LAB0)))
        (SETQ D25 (CDISA CC2 CC5))
        (COND
         ((*GREAT DD (*PLUS D14 (*PLUS D25 D36)))
          (RPLACD CL1 CL4)
          (RPLACD CL5 CL2)
          (RPLACD CL3 CL6)
          (GO LAB0)))
        (COND
         ((*GREAT DD (*PLUS D15 (*PLUS D26 D34)))
          (RPLACD CL1 CL5)
          (REVERSEXX CL2 CL5)
          (RPLACD CL2 CL6)
          (GO LAB0)))
        (SETQ D46 (CDISA CC4 CC6))
        (COND
         ((*GREAT DD (*PLUS D46 (*PLUS D12 D35)))
          (RPLACD CL3 CL5)
          (REVERSEXX CL4 CL5)
          (RPLACD CL4 CL6)
          (GO LAB0)))
        (COND
         ((*GREAT DD (*PLUS D13 (*PLUS D25 D46)))
          (RPLACD CL1 CL3)
          (REVERSEXX CL2 CL3)
          (RPLACD CL2 CL5)
          (REVERSEXX CL4 CL5)
          (RPLACD CL4 CL6)
          (GO LAB0)))
        (SETQ L3 (CDR L3))
        (GO LAB3)))
EXPR)

(DEFPROP CLSOL
 (LAMBDA(NAME)
  (PROG (CITIES IN OUT NUM L I COST J CC1 CC2)
        (COMMENT CRETES A SOLUTION TO PROBLEM AT HONT NAME)
        (COMMENT USING LINS THREEOPT PROCEDURE)
        (SETQ NAME (CAR (POINTER NAME)))
        (SETQ CITIES (GET NAME (QUOTE OBJECTSP)))
        (COND ((NULL CITIES) (RETURN NIL)))
        (COND
         ((EQ NAME (QUOTE UNIV))
          (SETQ IN (CAR CITIES))
          (SETQ OUT IN)
          (GO LAB))
         (T (CBPTS1 NAME)))
        (SETQ IN (CAAR (GET NAME (QUOTE STRUCTUREP))))
        (SETQ OUT (CADAR (GET NAME (QUOTE STRUCTUREP))))
        (SETQ CITIES (CONS IN (REMOVEX IN CITIES)))
   LAB  (COND
         ((EQ IN OUT)
          (SETQ CITIES (APPEND CITIES (LIST OUT))))
         (T
          (SETQ CITIES
                (APPEND (REMOVEX OUT CITIES) (LIST OUT)))))
```

```
(SETQ NUM (LENGTH CITIES))
(COND
 ((*LESS NUM 4) (GO LAB1))
 ((*GREAT NUM MAXLIN)
  (RETURN
   (QUOTE
    (MAXIMUM NUMBER
             OF
             CITIES
             EXCEED
             FOR
             LINSOLVE)))))
(COMMENT SET
         UP
         COPY
         OF
         DST
         ARRAY
         IN
         LOWER
         HALF
         OF
         DST)
(SETQ I 0)
(MAP
 (FUNCTION
  (LAMBDA(X)
   (PROG NIL
         (SETQ I (ADD1 I))
         (STORE (CLIST I) (CAR X))
         (SETQ J I)
         (MAPC
          (FUNCTION
           (LAMBDA(Y)
            (PROG (D)
                  (SETQ J (ADD1 J))
                  (COND
                   ((NULL
                     (AND
                      (GET (CAR X)
                           (QUOTE TERMINALP))
                      (GET Y (QUOTE TERMINALP))))
                    (SETQ D (DIS1 (CAR X) Y)))
                   (T (SETQ CC1 (EVAL (CAR X)))
                      (SETQ CC2 (EVAL Y))
                      (COND
                       ((*LESS CC1 CC2)
                        (SETQ D (DST CC1 CC2)))
                       (T (SETQ D (DST CC2 CC1))))))
                  (STORE (CDISA I J) D)
                  (STORE (CDISA J I) D)
                  (RETURN NIL))))
          (CDR X))
         (RETURN NIL))))
 CITIES)
(SETQ L (ILIST 1 NUM))
(COND ((NULL MLIN) (SETQ MLIN 2)))
(COND ((NULL RLIN) (SETQ RLIN 2)))
(SETQ L (LIN1 L))
(SETQ COST (COST1 L))
```

```
        (SETQ CITIES NIL)
    LAB2 (SETQ CITIES (CONS (CLIST (CAR L)) CITIES))
        (SETQ L (CDR L))
    LAB1 (COND
         ((NULL L)
          (PUTPROP NAME
                   (LIST (LIST IN OUT) (REVERSEX CITIES))
                   (QUOTE STRUCTUREP))
          (SETQ MLIN NIL)
          (SETQ RLIN NIL)
          (DSOL NAME)
          (RETURN COST)))
         (GO LAB2)))
EXPR)

(DEFPROP COSTL
 (LAMBDA(L)
  (PROG (COUNT X Y)
        (COMMENT USES
                 DST
                 ARRAY
                 TO
                 FIND
                 COST
                 OF
                 A
                 SOL
                 REPRESENTED)
        (COMMENT BY A LIST OF PERMUTED INTEGERS)
        (COND ((NULL L) (RETURN NIL)))
        (SETQ COUNT 0)
    LAB (COND ((NULL (CDR L)) (RETURN COUNT)))
        (SETQ X (CAR L))
        (SETQ Y (CADR L))
        (SETQ D (CDISA X Y))
        (COND
         ((MINUSP D) (SETQ D (DIS1 (CLIST X) (CLIST Y)))))
        (SETQ COUNT (*PLUS COUNT D))
        (SETQ L (CDR L))
        (GO LAB)))
EXPR)

(DEFPROP LIN1
 (LAMBDA(L)
  (PROG (INT I L1 P1 P2 P CC1 CC2 X Y Z)
        (COMMENT GOES
                 THRU
                 RLIN
                 REDUCTION
                 CYCLES
                 IN
                 FINDING
                 BEST
                 PERM)
        (COMMENT OF L IE CALLS LIN2 RLIN TIMES)
        (SETQ I 1)
    LAB3 (SETQ N (LENGTH L))
        (COND ((*LESS N 4) (GO LAB5)))
        (SETQ L (LIN2 L))
        (COND
```

```
        ((NEQ I RLIN)
         (COND ((NULL INT) (SETQ I (ADD1 I)) (GO LAB3))
               (T (GO LAB1))))))
   LAB5 (SETQ P1 L)
        (SETQ P2 (CDR L))
   LAB2 (SETQ L1
              (REVERSEX
               (GET (CLIST (CAR P1)) (QUOTE FOLLOWERSP))))
        (PUTPROP (CLIST (CAR P1)) NIL (QUOTE FOLLOWERSP))
        (COND ((NULL L1)
               (COND ((NULL P2) (RETURN L))
                     (T (SETQ P1 P2)
                        (SETQ P2 (CDR P1))
                        (GO LAB2))))
              (T (RPLACD P1 L1)
                 (COND ((NULL P2) (RETURN L))
                       (T (NCONC L1 P2)
                          (SETQ P1 (CDR P1))
                          (SETQ P2 (CDR P1))
                          (GO LAB2)))))
   LAB1 (SETQ P1 L)
        (SETQ P (CDR P1))
        (SETQ P2 (CDR P))
   LAB6 (COND
         ((NULL P2) (SETQ INT NIL)
                    (SETQ I (ADD1 I))
                    (GO LAB3)))
        (COND
         ((NMEMX (CAR P1) (CAR P) INT)
          (SETQ P1 P)
          (SETQ P P2)
          (SETQ P2 (CDR P2))
          (GO LAB6)))
   LAB4 (COND
         ((NMEMX (CAR P) (CAR P2) INT)
          (SETQ P1 P2)
          (SETQ P (CDR P1))
          (COND
           ((NULL P) (SETQ INT NIL)
                     (SETQ I (ADD1 I))
                     (GO LAB3)))
          (SETQ P2 (CDR P2))
          (GO LAB6)))
        (RPLACD P1 P2)
        (STORE (CDISA (CAR P1) (CAR P2)) -12221)
        (STORE (CDISA (CAR P2) (CAR P1)) -12221)
        (PUTPROP
         (CLIST (CAR P1))
         (CONS (CAR P)
               (GET (CLIST (CAR P1)) (QUOTE FOLLOWERSP)))
         (QUOTE FOLLOWERSP))
        (SETQ P P2)
        (SETQ P2 (CDR P2))
        (COND
         ((NULL P2) (SETQ INT NIL)
                    (SETQ I (ADD1 I))
                    (GO LAB3)))
        (GO LAB4)))
EXPR)
```

```
(DEFPROP LIN2
 (LAMBDA(L)
  (PROG (L1 S TEMP L2 COST1 COST2 I X Y L3)
        (COMMENT CREATES
                 MLIN
                 3
                 OPTS
                 OF
                 L
                 SETS
                 UP
                 REDUCTION
                 LIST)
        (COMMENT HAS THE ALMOST NTHREEOPT FEATURE)
        (SETQ I 1)
        (SETQ L1 (ECOPY L))
        (SETQ L2 (ECOPY L))
        (SETQ COST1 (COSTL L1))
        (SETQ L2 (NTHREEOPT (RPERM L2)))
        (SETQ COST2 (COSTL L2))
        (SETQ L3 L2)
   LAB1 (COND
         ((NOT (NULL (CDR L3)))
          (SETQ S (CONS (LIST (CAR L3) (CADR L3)) S))
          (SETQ INT (CONS (LIST (CAR L3) (CADR L3)) INT))
          (SETQ L3 (CDR L3))
          (GO LAB1)))
        (COND
         ((*LESS COST2 COST1)
          (SETQ X L1)
          (SETQ L1 L2)
          (SETQ L2 X)
          (SETQ COST1 COST2)))
   LAB3 (COND ((EQ I MLIN) (SETQ INT (REVERSEX INT))
                           (RETURN L1))
              (T (SETQ I (ADD1 I))))
        (SETQ L2 (NTHREEOPT (RPERM L2)))
        (SETQ COST2 (COSTL L2))
        (SETQ L3 L2)
   LAB2 (COND
         ((NOT (NULL (CDR L3)))
          (SETQ X (CAR L3))
          (SETQ Y (CADR L3))
          (COND
           ((NMEMX X Y S) (SETQ S (CONS (LIST X Y) S))))
          (COND
           ((NOT (NMEMX X Y INT))
            (SETQ TEMP (CONS (LIST X Y) TEMP))))
          (SETQ L3 (CDR L3))
          (GO LAB2)))
        (SETQ INT TEMP)
        (SETQ TEMP NIL)
        (COND
         ((*LESS COST2 COST1)
          (SETQ X L1)
          (SETQ L1 L2)
          (SETQ L2 X)
          (SETQ COST1 COST2)))
        (GO LAB3)))
EXPR)
```

```
(DEFPROP ECOPY
 (LAMBDA(L)
  (COND ((NULL L) NIL) (T (CONS (CAR L) (ECOPY (CDR L))))))
EXPR)

(DEFPROP ATHREEOPT
 (LAMBDA(L)
  (PROG (CC1 CC2
             CC3
             CC4
             CC5
             CC6
             MIN
             X
             OPTION
             L1
             L2
             L3
             D12
             D13
             D14
             D15
             D24
             D25
             D26
             D34
             D35
             D36
             D46
             D56)
        (COMMENT COMPS
                 A
                 3
                 Ø
                 OPT
                 PERM
                 OF
                 L
                 FREE
                 VAR
                 FLAG
                 TELLS
                 OF)
        (COMMENT CHANGES DESTROYS L WITH RPLACD'S)
        (COMMENT USES S TO IGNORE CERTAIN POSSIBILITIES)
   LABØ (SETQ L1 L)
   LAB1 (COND ((NULL (CDR L1)) (RETURN L)))
        (SETQ CC1 L1)
        (SETQ CC2 (CDR L1))
        (COND
         ((MEMBER (LIST (CAR CC1) (CAR CC2)) S)
          (SETQ L1 (CDR L1))
          (GO LAB1)))
        (SETQ D12 (CDISA (CAR CC1) (CAR CC2)))
        (COND ((MINUSP D12) (SETQ L1 (CDR L1)) (GO LAB1)))
        (SETQ L2 (CDR L1))
   LAB2 (COND
         ((NULL (CDR L2)) (SETQ L1 (CDR L1)) (GO LAB1)))
        (SETQ CC3 L2)
```

```
        (SETQ CC4 (CDR L2))
        (SETQ D34 (CDISA (CAR CC3) (CAR CC4)))
        (COND ((MINUSP D34) (SETQ L2 (CDR L2)) (GO LAB2)))
        (SETQ D13 (CDISA (CAR CC1) (CAR CC3)))
        (SETQ D14 (CDISA (CAR CC1) (CAR CC4)))
        (SETQ D24 (CDISA (CAR CC2) (CAR CC4)))
        (SETQ L3 (CDR L2))
LAB3    (COND
         ((NULL (CDR L3)) (SETQ L2 (CDR L2)) (GO LAB2)))
        (SETQ CC5 L3)
        (SETQ CC6 (CDR L3))
        (SETQ D56 (CDISA (CAR CC5) (CAR CC6)))
        (COND ((MINUSP D56) (SETQ L3 (CDR L3)) (GO LAB3)))
        (SETQ DD (*PLUS (*PLUS D12 D34) D56))
        (SETQ D25 (CDISA (CAR CC2) (CAR CC5)))
        (SETQ D46 (CDISA (CAR CC4) (CAR CC6)))
        (SETQ MIN (PLUS D13 D25 D46))
        (SETQ OPTION 1)
        (SETQ D26 (CDISA (CAR CC2) (CAR CC6)))
        (SETQ D35 (CDISA (CAR CC3) (CAR CC5)))
        (SETQ X (PLUS D14 D26 D35))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 2)))
        (SETQ D15 (CDISA (CAR CC1) (CAR CC5)))
        (SETQ D36 (CDISA (CAR CC3) (CAR CC6)))
        (SETQ X (PLUS D15 D24 D36))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 3)))
        (SETQ X (PLUS D14 D25 D36))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 4)))
        (SETQ X (PLUS D12 D35 D46))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 5)))
        (SETQ X (PLUS D15 D26 D34))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 6)))
        (SETQ X (PLUS D13 D24 D56))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 7)))
        (COND
         ((NOT (*LESS MIN DD))
          (SETQ L3 (CDR L3))
          (GO LAB3)))
        (SETQ FLAG T)
        (COND ((EQ OPTION 1) (RPLACD CC1 CC3)
                             (REVERSEXX CC2 CC3)
                             (RPLACD CC2 CC5)
                             (REVERSEXX CC4 CC5)
                             (RPLACD CC4 CC6))
              ((EQ OPTION 2) (RPLACD CC1 CC4)
                             (REVERSEXX CC2 CC3)
                             (RPLACD CC2 CC6)
                             (RPLACD CC5 CC3))
              ((EQ OPTION 3) (RPLACD CC1 CC5)
                             (REVERSEXX CC4 CC5)
                             (RPLACD CC4 CC2)
                             (RPLACD CC3 CC6))
              ((EQ OPTION 4) (RPLACD CC1 CC4)
                             (RPLACD CC5 CC2)
                             (RPLACD CC3 CC6))
              ((EQ OPTION 5) (RPLACD CC3 CC5)
                             (REVERSEXX CC4 CC5)
                             (RPLACD CC4 CC6))
              ((EQ OPTION 6) (RPLACD CC1 CC5)
                             (REVERSEXX CC2 CC5)
```

```
                                (RPLACD CC2 CC6))
                ((EQ OPTION 7) (RPLACD CC1 CC3)
                                (REVERSEXX CC2 CC3)
                                (RPLACD CC2 CC4)))
        (GO LAB0)))
EXPR)

(DEFPROP REVERSEXX
 (LAMBDA(P1 P2)
  (PROG (X1 X X2)
        (COMMENT GOES
                 IN
                 AND
                 REVERSES
                 PART
                 OF
                 A
                 LIST
                 FROM
                 P1
                 TO
                 P2
                 INCLUSIVE)
        (COMMENT DESTROYS LINKS WITH REST OF LIST)
        (SETQ X1 NIL)
        (SETQ X P1)
   LAB  (SETQ X2 (CDR X))
        (RPLACD X X1)
        (COND ((EQ X P2) (RETURN P2)))
        (SETQ X1 X)
        (SETQ X X2)
        (GO LAB)))
EXPR)

(DEFPROP THREEOPT
 (LAMBDA(L)
  (PROG (CC1 CC2
             CC3
             CC4
             CC5
             CC6
             MIN
             X
             OPTION
             L1
             L2
             L3
             D12
             D13
             D14
             D15
             D24
             D25
             D26
             D34
             D35
             D36
             D46
             D56)
        (COMMENT COMPS
```

```
                A
                3
                Ø
                OPT
                PERM
                OF
                L
                FREE
                VAR
                FLAG
                TELLS
                OF)
        (COMMENT CHANGES DESTROYS L WITH RPLACD'S)
LABØ    (SETQ L1 L)
LAB1    (COND ((NULL (CDR L1)) (RETURN L)))
        (SETQ CC1 L1)
        (SETQ CC2 (CDR L1))
        (SETQ D12 (CDISA (CAR CC1) (CAR CC2)))
        (COND ((MINUSP D12) (SETQ L1 (CDR L1)) (GO LAB1)))
        (SETQ L2 (CDR L1))
LAB2    (COND
         ((NULL (CDR L2)) (SETQ L1 (CDR L1)) (GO LAB1)))
        (SETQ CC3 L2)
        (SETQ CC4 (CDR L2))
        (SETQ D34 (CDISA (CAR CC3) (CAR CC4)))
        (COND ((MINUSP D34) (SETQ L2 (CDR L2)) (GO LAB2)))
        (SETQ D13 (CDISA (CAR CC1) (CAR CC3)))
        (SETQ D14 (CDISA (CAR CC1) (CAR CC4)))
        (SETQ D24 (CDISA (CAR CC2) (CAR CC4)))
        (SETQ L3 (CDR L2))
LAB3    (COND
         ((NULL (CDR L3)) (SETQ L2 (CDR L2)) (GO LAB2)))
        (SETQ CC5 L3)
        (SETQ CC6 (CDR L3))
        (SETQ D56 (CDISA (CAR CC5) (CAR CC6)))
        (COND ((MINUSP D56) (SETQ L3 (CDR L3)) (GO LAB3)))
        (SETQ DD (*PLUS (*PLUS D12 D34) D56))
        (SETQ D25 (CDISA (CAR CC2) (CAR CC5)))
        (SETQ D46 (CDISA (CAR CC4) (CAR CC6)))
        (SETQ MIN (PLUS D13 D25 D46))
        (SETQ OPTION 1)
        (SETQ D26 (CDISA (CAR CC2) (CAR CC6)))
        (SETQ D35 (CDISA (CAR CC3) (CAR CC5)))
        (SETQ X (PLUS D14 D26 D35))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 2)))
        (SETQ D15 (CDISA (CAR CC1) (CAR CC5)))
        (SETQ D36 (CDISA (CAR CC3) (CAR CC6)))
        (SETQ X (PLUS D15 D24 D36))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 3)))
        (SETQ X (PLUS D14 D25 D36))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 4)))
        (SETQ X (PLUS D12 D35 D46))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 5)))
        (SETQ X (PLUS D15 D26 D34))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 6)))
        (SETQ X (PLUS D13 D24 D56))
        (COND ((*LESS X MIN) (SETQ MIN X) (SETQ OPTION 7)))
        (COND
         ((NOT (*LESS MIN DD))
          (SETQ L3 (CDR L3))
```

```
        (SETQ P1 L)
        (SETQ P (CDR L))
        (SETQ P2 (CDR P))
        (SETQ I 2)
   LAB2 (COND ((EQ I R)
               (RPLACD P1 P2)
               (RPLACD P4 P)
               (SETQ P4 P)
               (SETQ N (SUB1 N))
               (COND
                ((EQ N 2) (RPLACD P4 (CDR L))
                          (RPLACD L (CDR P3))
                          (RETURN L)))
               (GO LAB1))
              (T (SETQ I (ADD1 I))
                 (SETQ P1 P)
                 (SETQ P P2)
                 (SETQ P2 (CDR P2))
                 (GO LAB2))))
EXPR)

(DEFPROP RND
 (LAMBDA NIL
  (PROG NIL
        (SETQ SEED
              (REMAINDER (*PLUS (*TIMES 2011 SEED) 1)
                         40000))
        (RETURN (*QUO (*PLUS SEED 0,0) 40000))))
EXPR)

(DEFPROP LIN
 (LIN NMEMX
      NTHREEOPT
      CLSOL
      COSTL
      LIN1
      LIN2
      ECOPY
      ATHREEOPT
      REVERSEXX
      THREEOPT
      RANDOMI
      REVERSEX
      RPERM
      RND
      LIN)
VALUE)
```

```
          (GO LAB3)))
        (SETQ FLAG T)
        (COND ((EQ OPTION 1) (RPLACD CC1 CC3)
                             (REVERSEXX CC2 CC3)
                             (RPLACD CC2 CC5)
                             (REVERSEXX CC4 CC5)
                             (RPLACD CC4 CC6))
              ((EQ OPTION 2) (RPLACD CC1 CC4)
                             (REVERSEXX CC2 CC3)
                             (RPLACD CC2 CC6)
                             (RPLACD CC5 CC3))
              ((EQ OPTION 3) (RPLACD CC1 CC5)
                             (REVERSEXX CC4 CC5)
                             (RPLACD CC4 CC2)
                             (RPLACD CC3 CC6))
              ((EQ OPTION 4) (RPLACD CC1 CC4)
                             (RPLACD CC5 CC2)
                             (RPLACD CC3 CC6))
              ((EQ OPTION 5) (RPLACD CC3 CC5)
                             (REVERSEXX CC4 CC5)
                             (RPLACD CC4 CC6))
              ((EQ OPTION 6) (RPLACD CC1 CC5)
                             (REVERSEXX CC2 CC5)
                             (RPLACD CC2 CC6))
              ((EQ OPTION 7) (RPLACD CC1 CC3)
                             (REVERSEXX CC2 CC3)
                             (RPLACD CC2 CC4)))
        (GO LAB0)))
EXPR)

(DEFPROP RANDOMI
 (LAMBDA(X Y)
  (FIX (*PLUS X (*TIMES (*DIF (ADD1 Y) X) (RND))))))
EXPR)

(DEFPROP REVERSEX
 (LAMBDA(L)
  (PROG (P1 P P2)
        (COMMENT REVERSES A LIST)
        (COND ((NULL L) (RETURN L)))
        (SETQ R1 NIL)
        (SETQ P L)
        (SETQ P2 (CDR L))
   LAB  (RPLACD P P1)
        (COND ((NULL P2) (RETURN P)))
        (SETQ P1 P)
        (SETQ P P2)
        (SETQ P2 (CDR P2))
        (GO LAB)))
EXPR)

(DEFPROP RPERM
 (LAMBDA(L)
  (PROG (R RL P1 P P2 P3 P4 N I)
        (COMMENT RANDOM PERMUTES L DESTROYS L L NON EMPTY)
        (COMMENT RETAINS FIRST AND LAST ELEMENTS)
        (SETQ N (LENGTH L))
        (SETQ P3 (LIST (QUOTE DUM)))
        (SETQ P4 P3)
   LAB1 (SETQ R (RANDOMI 2 (SUB1 N)))
```

```
(DEFPROP CROES
 (CROES CROES CROES1 CROES2 INVERSION CUSTC DIS2 CCSOL)
VALUE)

(DEFPROP CROES
 (CROES CROES CROES1 CROES2 INVERSION CUSTC DIS2 CCSOL)
VALUE)

(DEFPROP CROES1
 (LAMBDA(L)
  (PROG (INT I L1 P1 P2 P CC1 CC2 X Y Z)
        (COMMENT GOES
                 THRU
                 RCROES
                 REDUCTION
                 CYCLES
                 IN
                 ENDING
                 BEST
                 PERM)
        (COMMENT OF L IE CALLS CROES2 RCROES TIMES)
        (SETQ I 1)
   LAB3 (SETQ N (LENGTH L))
        (COND ((*LESS N 4) (GO LAB5)))
        (SETQ L (CROES2 L))
        (COND
         ((NEQ I RCROES)
          (COND ((NULL INT) (SETQ I (ADD1 I)) (GO LAB3))
                (T (GO LAB1)))))
   LAB5 (SETQ P1 L)
        (SETQ P2 (CDR L))
   LAB2 (SETQ L1
              (REVERSEX
               (GET (CLIST (CAR P1)) (QUOTE FOLLOWERSP))))
        (PUTPROP (CLIST (CAR P1)) NIL (QUOTE FOLLOWERSP))
        (COND ((NULL L1)
               (COND ((NULL P2) (RETURN L))
                     (T (SETQ P1 P2)
                        (SETQ P2 (CDR P1))
                        (GO LAB2))))
              (T (RPLACD P1 L1)
                 (COND ((NULL P2) (RETURN L))
                       (T (NCONC L1 P2)
                          (SETQ P1 (CDR P1))
                          (SETQ P2 (CDR P1))
                          (GO LAB2)))))
   LAB1 (SETQ P1 L)
        (SETQ P (CDR P1))
        (SETQ P2 (CDR P))
   LAB6 (COND
         ((NULL P2) (SETQ INT NIL)
                    (SETQ I (ADD1 I))
                    (GO LAB3)))
        (COND
         ((NUMEMX (CAR P1) (CAR P) INT)
          (SETQ P1 P)
          (SETQ P P2)
          (SETQ P2 (CDR P2))
```

```
          (GO LAB6)))
   LAB4 (COND
         ((NMEMX (CAR P) (CAR P2) INT)
          (SETQ P1 P2)
          (SETQ P (CDR P1))
          (COND
           ((NULL P) (SETQ INT NIL)
                     (SETQ I (ADD1 I))
                     (GO LAB3)))
          (SETQ P2 (CDR P2))
          (GO LAB6)))
        (RPLACD P1 P2)
        (STORE (CDISA (CAR P1) (CAR P2)) -12221)
        (STORE (CDISA (CAR P2) (CAR P1)) -12221)
        (PUTPROP
         (CLIST (CAR P1))
         (CONS (CAR P)
               (GET (CLIST (CAR P1)) (QUOTE FOLLOWERSP)))
         (QUOTE FOLLOWERSP))
        (SETQ P P2)
        (SETQ P2 (CDR P2))
        (COND
         ((NULL P2) (SETQ INT NIL)
                    (SETQ I (ADD1 I))
                    (GO LAB3)))
        (GO LAB4)))
EXPR)

(DEFPROP CROES2
 (LAMBDA(L)
  (PROG (L1 S TEMP L2 COST1 COST2 I X Y L3)
        (COMMENT CREATES
                 MCROES
                 INVERSIONS
                 OF
                 L
                 SETS
                 UP
                 REDUCTION
                 LIST)
        (COMMENT HAS THE ALMOST INVERSIONS FEATURE)
        (SETQ I 1)
        (SETQ L1 (ECOPY L))
        (SETQ L2 (ECOPY L))
        (SETQ COST1 (COSTC L1))
        (SETQ L2 (INVERSION (RPERM L2)))
        (SETQ COST2 (COSTC L2))
        (SETQ L3 L2)
   LAB1 (COND
         ((NOT (NULL (CDR L3)))
          (SETQ S (CONS (LIST (CAR L3) (CADR L3)) S))
          (SETQ INT (CONS (LIST (CAR L3) (CADR L3)) INT))
          (SETQ L3 (CDR L3))
          (GO LAB1)))
        (COND
         ((*LESS COST2 COST1)
          (SETQ X L1)
          (SETQ L1 L2)
          (SETQ L2 X)
          (SETQ COST1 COST2)))
```

```
    LAB3 (COND ((EQ I MCROES) (SETQ INT (REVERSEX INT))
                              (RETURN L1))
               (T (SETQ I (ADD1 I))))
         (SETQ L2 (INVERSION (RPERM L2)))
         (SETQ COST2 (COSTC L2))
         (SETQ L3 L2)
    LAB2 (COND
          ((NOT (NULL (CDR L3)))
           (SETQ X (CAR L3))
           (SETQ Y (CADR L3))
           (COND
            ((NMEMX X Y S) (SETQ S (CONS (LIST X Y) S))))
           (COND
            ((NOT (NMEMX X Y INT))
             (SETQ TEMP (CONS (LIST X Y) TEMP))))
           (SETQ L3 (CDR L3))
           (GO LAB2)))
         (SETQ INT TEMP)
         (SETQ TEMP NIL)
         (COND
          ((*LESS COST2 COST1)
           (SETQ X L1)
           (SETQ L1 L2)
           (SETQ L2 X)
           (SETQ COST1 COST2)))
         (GO LAB3)))
EXPR)

(DEFPROP INVERSION
 (LAMBDA(L)
  (PROG (L1 L2 CC1 CC2 CC3 CC4 D12 D34)
        (COMMENT COMPUTES AN INVERSION PERM OF L DESTROYS L)
    LAB0 (SETQ L1 L)
    LAB1 (COND ((NULL (CDR L1)) (RETURN L)))
         (SETQ CC1 L1)
         (SETQ CC2 (CDR L1))
         (SETQ D12 (DIS2 (CAR CC1) (CAR CC2)))
         (COND ((MINUSP D12) (SETQ L1 (CDR L1)) (GO LAB1)))
         (SETQ L2 (CDR L1))
    LAB2 (COND
          ((NULL (CDR L2)) (SETQ L1 (CDR L1)) (GO LAB1)))
         (SETQ CC3 L2)
         (SETQ CC4 (CDR L2))
         (SETQ D34 (DIS2 (CAR CC3) (CAR CC4)))
         (COND ((MINUSP D34) (SETQ L2 (CDR L2)) (GO LAB2))))
         (COND ((*LESS (*PLUS (DIS2 (CAR CC1) (CAR CC3))
                              (DIS2 (CAR CC2) (CAR CC4)))
                       (*PLUS D12 D34))
                (RPLACD CC1 CC3)
                (REVERSEXY CC2 CC3)
                (RPLACD CC2 CC4)
                (GO LAB2))
               (T (SETQ L2 (CDR L2)) (GO LAB2)))))
EXPR)

(DEFPROP COSTC
 (LAMBDA(L)
  (PROG (COUNT X Y)
        (COMMENT USES
                 DST
```

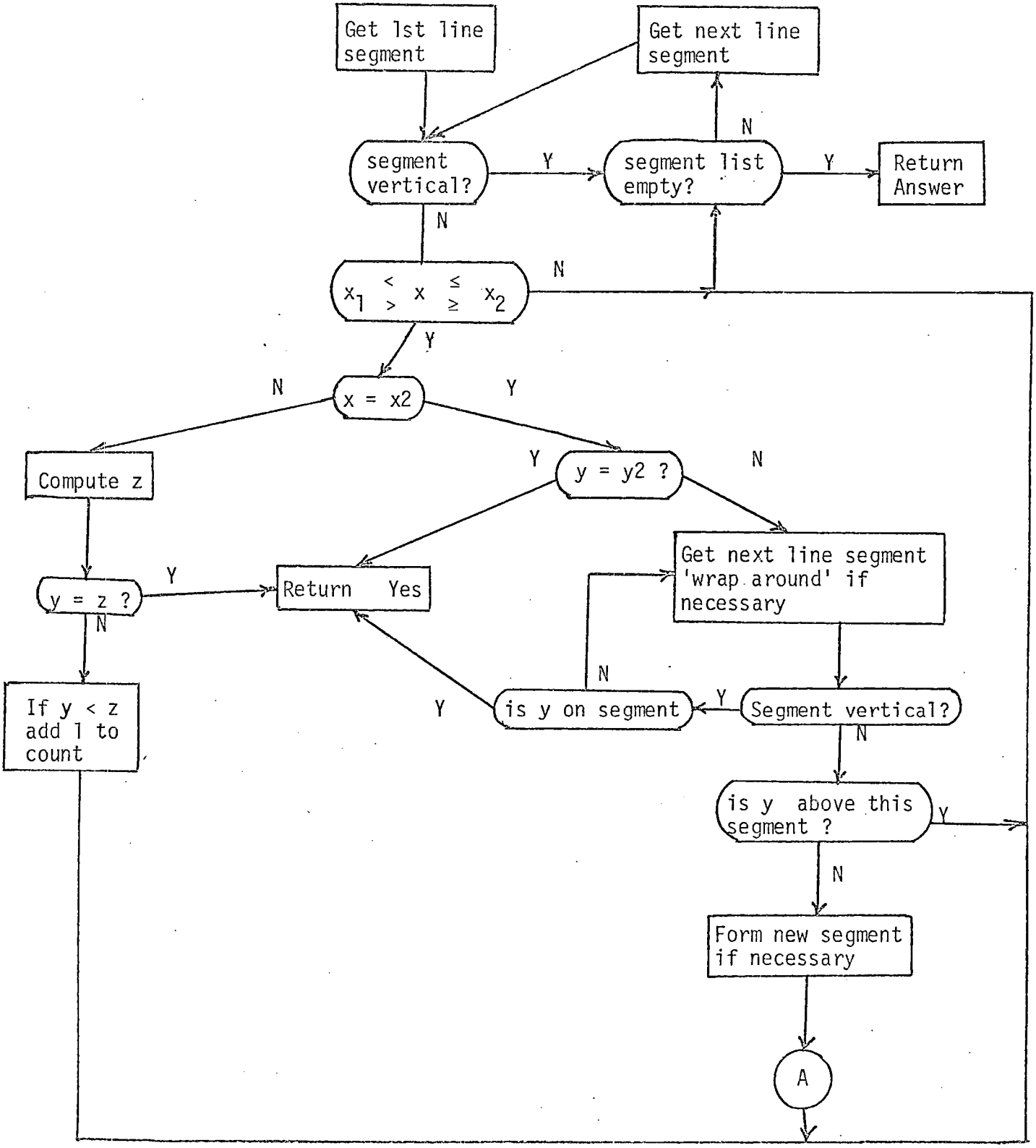
```
                    ARRAY
                    TO
                    FIND
                    COST
                    OF
                    A
                    SOL
                    REPRESENTED)
          (COMMENT BY A LIST OF PERMUTED INTEGERS)
          (COND ((NULL L) (RETURN NIL)))
          (SETQ COUNT 0)
    LAB   (COND ((NULL (CDR L)) (RETURN COUNT)))
          (SETQ X (CAR L))
          (SETQ Y (CADR L))
          (COND ((*LESS X Y) (SETQ D (DST Y X)))
                (T (SETQ D (DST X Y))))
          (COND
           ((MINUSP D) (SETQ D (DIS1 (CLIST X) (CLIST Y)))))
          (SETQ COUNT (*PLUS COUNT D))
          (SETQ L (CDR L))
          (GO LAB)))
EXPR)

(DEFPROP DIS2
 (LAMBDA(CC1 CC2)
  (COND ((*LESS CC1 CC2) (DST CC2 CC1)) (T (DST CC1 CC2))))
EXPR)

(DEFPROP CCSOL
 (LAMBDA(NAME)
  (PROG (CITIES IN OUT NUM L I COST J CC1 CC2)
          (COMMENT CRETES A SOLUTION TO PROBLEM AT NONT NAME)
          (COMMENT USING CROES INVERSION PROCEDURE)
          (SETQ NAME (CAR (POINTER NAME)))
          (SETQ CITIES (GET NAME (QUOTE OBJECTSP)))
          (COND ((NULL CITIES) (RETURN NIL)))
          (COND
           ((EQ NAME (QUOTE UNIV))
            (SETQ IN (CAR CITIES))
            (SETQ OUT IN)
            (GO LAB))
           (T (CBPTS1 NAME)))
          (SETQ IN (CAAR (GET NAME (QUOTE STRUCTUREP))))
          (SETQ OUT (CADAR (GET NAME (QUOTE STRUCTUREP))))
          (SETQ CITIES (CONS IN (REMOVEX IN CITIES)))
    LAB   (COND
           ((EQ IN OUT)
            (SETQ CITIES (APPEND CITIES (LIST OUT))))
           (T
            (SETQ CITIES
                  (APPEND (REMOVEX OUT CITIES) (LIST OUT)))))
          (SETQ NUM (LENGTH CITIES))
          (COND
           ((*LESS NUM 4) (GO LAB1))
           ((*GREAT NUM MAXCROES)
            (RETURN
             (QUOTE
              (MAXIMUM NUMBER
                       OF
                       CITIES
```

```
                    EXCEED
                    FOR
                    CROESSOLVE)))))
        (COMMENT SET
                 UP
                 COPY
                 OF
                 DST
                 ARRAY
                 IN
                 LOWER
                 HALF
                 OF
                 DST)
        (SETQ I 0)
        (MAP
         (FUNCTION
          (LAMBDA(X)
           (PROG NIL
                 (SETQ I (ADD1 I))
                 (STORE (CLIST I) (CAR X))
                 (SETQ J I)
                 (MAPC
                  (FUNCTION
                   (LAMBDA(Y)
                    (PROG (D)
                          (SETQ J (ADD1 J))
                          (COND
                           ((NULL
                             (AND
                              (GET (CAR X)
                                   (QUOTE TERMINALP))
                              (GET Y (QUOTE TERMINALP))))
                            (SETQ D (DIS1 (CAR X) Y)))
                           (T (SETQ CC1 (EVAL (CAR X)))
                              (SETQ CC2 (EVAL Y))
                              (COND
                               ((*LESS CC1 CC2)
                                (SETQ D (DST CC1 CC2)))
                               (T (SETQ D (DST CC2 CC1))))))
                          (STORE (DST J I) D)
                          (RETURN NIL))))
                  (CDR X))
                 (RETURN NIL))))
         CITIES)
        (SETQ L (ILIST 1 NUM))
        (COND ((NULL MCROES) (SETQ MCROES 2)))
        (COND ((NULL RCROES) (SETQ RCROES 2)))
        (SETQ L (CROES1 L))
        (SETQ COST (COST0 L))
        (SETQ CITIES NIL)
LAB2    (SETQ CITIES (CONS (CLIST (CAR L)) CITIES))
        (SETQ L (CDR L))
LAB1    (COND
         ((NULL L)
          (PUTPROP NAME
                   (LIST (LIST IN OUT) (REVERSEX CITIES))
                   (QUOTE STRUCTURE))
          (SETQ MCROES NIL)
          (SETQ RCROES NIL)
```

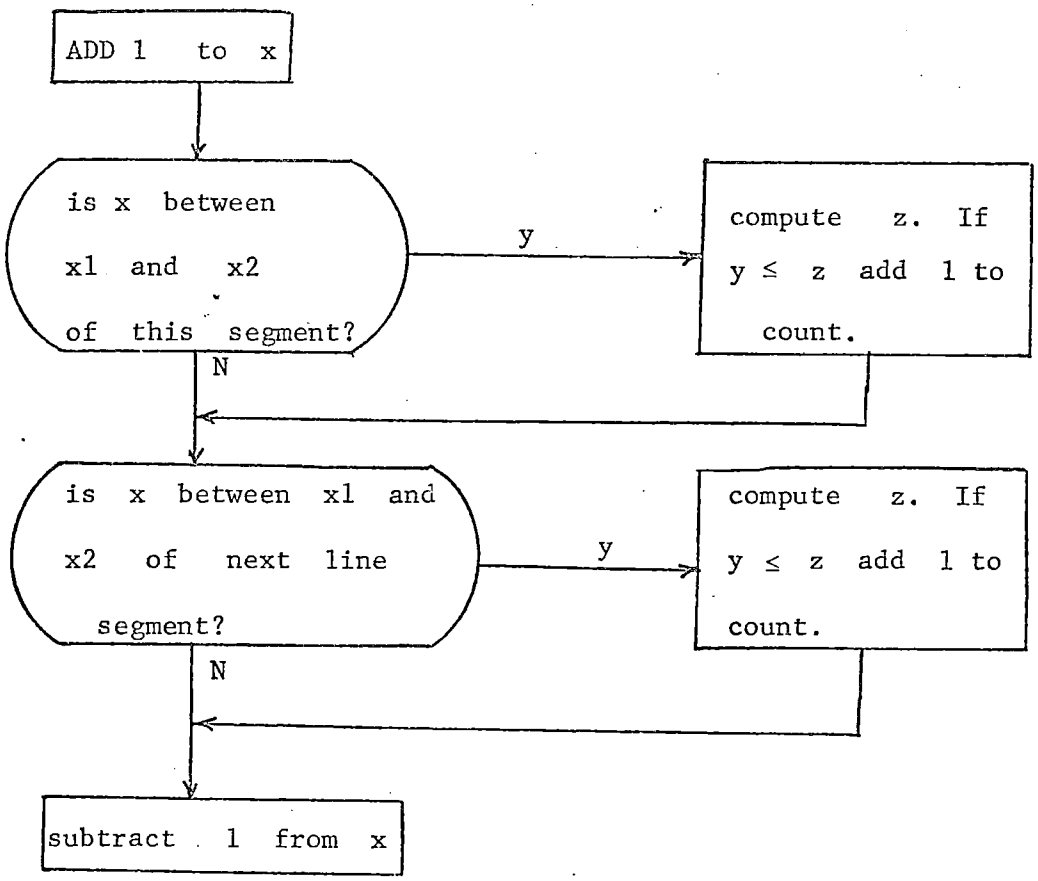
```
        (DSQL NAME)
        (RETURN COST)))
    (GO LAB2)))
EXPR)
```

Appendix V System Commands and Solution and Display Utility Routines

Flow Chart for the Interior Algorithm (checks if a point is inside a polygon)



Subroutine A:



```
(DEFPROP FUDGE
 (LAMBDA(NAME)
  (COND ((NULL (CADR (GET NAME (QUOTE STRUCTUREP)))) NAME)
        (T
         (FUDGE
          (CAR
           (LAST (CADR (GET NAME (QUOTE STRUCTUREP))))))))))
EXPR)

(DEFPROP ECITIES
 (LAMBDA NIL (PROG NIL (SELECT 1) (CLEAR)))
EXPR)

(DEFPROP DSYNTHS
 (LAMBDA NIL
  (PROG (L)
        (ESYNTHS)
        (SETQ L (LIST (QUOTE UNIV)))
   LAB  (DSUB (CAR L))
        (SETQ L (APPEND L (GET (CAR L) (QUOTE OBJECTSP))))
        (SETQ L (CDR L))
        (COND ((NULL L) (RETURN T)) (T (GO LAB))))))
EXPR)

(DEFPROP ECITYNAMES
 (LAMBDA NIL (PROG NIL (SELECT 3) (CLEAR) (RETURN NIL)))
EXPR)

(DEFPROP DCITYNAMES
 (LAMBDA NIL
  (PROG (C P CS)
        (COMMENT DISPLAYS THE CITY PRINT NAMES)
        (SELECT 3)
        (COND ((NULL CITIES) (RETURN NIL)))
        (SETQ CS CNAMES)
   LAB  (SETQ C (CAR CS))
        (SETQ P (GET C (QUOTE CENTROIDP)))
        (TEXT C (CAR P) (CADR P))
        (SETQ CS (CDR CS))
        (COND ((NULL CS) (RETURN NIL)) (T (GO LAB))))))
EXPR)

(DEFPROP ESUBNAMES
 (LAMBDA NIL (PROG NIL (SELECT 3) (CLEAR) (RETURN NIL)))
EXPR)

(DEFPROP EGARBAGE
 (LAMBDA NIL
  (PROG NIL
        (SELECT 3)
        (CLEAR)
        (SELECT 4)
        (CLEAR)
        (SELECT 7)
        (CLEAR)))
EXPR)

(DEFPROP CPPTS1
 (LAMBDA(NOUT)
  (PROG (P1 P2 X)
```

```
        (COMMENT CREATES TWO BDYPTS FOR GIVEN NONT)
        (PRINT (QUOTE (POINTER TO FIRST BOUNDARY POINT?)))
        (SETQ X (GET (QUOTE STATUS) (QUOTE CONTEXT)))
        (PUTPROP (QUOTE STATUS) NONT (QUOTE CONTEXT))
        (SETQ P1 (CAR (POINTER (QUOTE ***))))
        (PRINT (QUOTE (POINTER TO SECOND BOUNDARY POINT?)))
        (SETQ P2 (CAR (POINTER (QUOTE ***))))
        (PUTPROP (QUOTE STATUS) X (QUOTE CONTEXT))
        (PUTPROP
         NONT
         (LIST (LIST P1 P2)
               (CADR (GET NONT (QUOTE STRUCTUREP))))
         (QUOTE STRUCTUREP))
        (RETURN NONT)))
EXPR)

(DEFPROP DSOLSEX
 (LAMBDA(NAME)
  (PROG (L)
        (COMMENT DISPLAYS
                 ALL
                 SOLUTIONS
                 EXCEPT
                 THOSE
                 ROOTED
                 AT
                 NAME)
        (SETQ NAME (CAR (POINTER NAME)))
        (SELECT 5)
        (CLEAR)
        (SETQ L (LIST (QUOTE UNIV)))
   LAB  (COND ((NULL L) (RETURN NIL)))
        (COND ((EQ (CAR L) NAME) (SETQ L (CDR L)) (GO LAB))
              (T
               (SETQ L
                     (APPEND
                      L
                      (GET (CAR L) (QUOTE OBJECTSP))))))
        (DSOL (CAR L))
        (SETQ L (CDR L))
        (GO LAB)))
EXPR)

(DEFPROP DSURNAMES
 (LAMBDA NIL
  (PROG (L)
        (SETQ L (LIST (QUOTE UNIV)))
   LAB  (DSURNAME (CAR L))
        (SETQ L (APPEND L (GET (CAR L) (QUOTE OBJECTSP))))
        (SETQ L (CDR L))
        (COND ((NULL L) (RETURN T)) (T (GO LAB)))))
EXPR)

(DEFPROP DSURNAME
 (LAMBDA(NAME)
  (PROG (X)
        (COMMENT DISPLAYS
                 THE
                 PRINT
                 NAME
```

```
                    OF
                    A
                    SUBPROBLEM-NONTERMINAL)
          (SETQ NAME (CAR (POINTER NAME)))
          (COND
           ((NULL (GET NAME (QUOTE OBJECTSP))) (RETURN NIL)))
          (SETQ X (GET NAME (QUOTE INTENTIONP)))
          (COND ((NULL X) (RETURN NIL)))
          (SELECT 3)
          (TEXT NAME (CAAR X) (CADAR X))
          (RETURN T)))
EXPR)

(DEFPROP DCITIES
 (LAMBDA NIL
  (PROG (CS P)
        (SETQ CS CITIES)
        (SELECT 1)
        (CLEAR)
   LAB  (COND ((NULL CS) (RETURN NIL)))
        (SETQ P (CAR CS))
        (POINT (CAR P) (CADR P))
        (SETQ CS (CDR CS))
        (GO LAB)))
EXPR)

(DEFPROP DCNAMES
 (LAMBDA NIL
  (PROG (C P CS)
        (COMMENT DISPLAYS THE CITY PRINT NAMES)
        (SELECT 3)
        (COND ((NULL CITIES) (RETURN NIL)))
        (SETQ CS CITYNAMES)
   LAB  (SETQ C (CAR CS))
        (SETQ P (GET C (QUOTE CENTROIDP)))
        (TEXT C (CAR P) (CADR P))
        (SETQ CS (CDR CS))
        (COND ((NULL CS) (RETURN NIL)) (T (GO LAB)))))
EXPR)

(DEFPROP OUT
 (LAMBDA(X)
  (PROG (B L)
        (COMMENT DETERMINES
                 THE
                 LOWEST
                 LEVEL
                 CENTROID
                 IN
                 A
                 SUB
                 WHICH)
        (COMMENT COMES AFTER X)
        (SETQ B (GET X (QUOTE BELONGSP)))
        (SETQ L (CADR (GET B (QUOTE STRUCTUREP))))
   LAB  (COND
         ((EQ (CAR L) X)
          (COND ((NULL (CDR L))
                 (COND ((EQ NAME B) (RETURN NIL))
                       (T (RETURN (OUT B))))))
```

```
            (T
             (RETURN
              (GET (CADR L) (QUOTE CENTROIDP))))))))
      (SETQ L (CDR L))
      (GO LAB)))

PR)

EFPROP SLAST
LAMBDA(L)
(COND ((NULL (CDR L)) NIL)
      ((NULL (CDDR L)) (CAR L))
      (T (SLAST (CDR L)))))

XPR)

DEFPROP IN
(LAMBDA(X)
 (PROG (B L)
       (COMMENT DETERMINES
                THE
                LOWEST
                LEVEL
                CENTROID
                INA
                SUB)
       (COMMENT WHICH COMES BEFORE X)
       (SETQ B (GET X (QUOTE BELONGSP)))
       (SETQ L (CADR (GET B (QUOTE STRUCTUREP))))
       (COND
        ((EQ X (CAR L))
         (COND ((EQ B (QUOTE UNIV))
                (RETURN (GET (SLAST L) (QUOTE CENTROIDP))))
               ((EQ B NAME) (RETURN NIL))
               (T (RETURN (IN B))))))
   LAB (COND ((EQ X (CADR L))
              (RETURN (GET (CAR L) (QUOTE CENTROIDP))))
             (T (SETQ L (CDR L)) (GO LAB)))))

EXPR)

(DEFPROP SYNTH2
 (LAMBDA(NAME)
  (PROG (X S P1 P2 K1 K2 ENTR EXX)
        (COMMENT GOES
                 THRU
                 SUBPROB
                 TREE
                 REVERSING
                 SUBSOLS
                 ON
                 BASIS)
        (COMMENT OF 'BEST FIT' HEURISTIC)
        (SETQ NAME (CAR (POINTER NAME)))
        (SETQ L (CADR (GET NAME (QUOTE STRUCTUREP))))
   LAB  (COND ((NULL L) (RETURN NIL)))
        (SETQ X (CAR L))
        (SETQ S (CADR (GET X (QUOTE STRUCTUREP))))
        (COND
         ((NEQ NIL S)
          (SETQ L (APPEND L S))
          (SETQ P1 (IN X))
          (SETQ P2 (OUT X))
```

```
          (COND
           ((AND (NULL P1) (NULL P2))
            (SETQ L (CDR L))
            (GO LAB)))
          (SETQ ENTR (CAAR (GET X (QUOTE STRUCTUREP))))
          (SETQ EXX (CADAR (GET X (QUOTE STRUCTUREP))))
          (COND
           ((NULL P1)
            (SETQ
             K1
             (DISTANCE1 P2 (GET EXX (QUOTE CENTROIDP))))
            (SETQ
             K2
             (DISTANCE1 P2 (GET ENTR (QUOTE CENTROIDP)))))
           (T
            (COND
             ((NULL P2)
              (SETQ
               K1
               (DISTANCE1 P1 (GET ENTR (QUOTE CENTROIDP))))
              (SETQ
               K2
               (DISTANCE1 P1 (GET EXX (QUOTE CENTROIDP)))))
             (T (SETQ
                 K1
                 (*PLUS
                  (DISTANCE1 P1
                             (GET ENTR (QUOTE CENTROIDP)))
                  (DISTANCE1 P2
                             (GET EXX (QUOTE CENTROIDP)))))
                (SETQ
                 K2
                 (*PLUS
                  (DISTANCE1 P1 (GET EXX (QUOTE CENTROIDP)))
                  (DISTANCE1
                   P2
                   (GET ENTR (QUOTE CENTROIDP))))))))))
          (COND
           ((LESSP K2 K1)
            (PUTPROP X
                     (LIST (LIST EXX ENTR) (REVERSE S))
                     (QUOTE STRUCTUREP)))))
          (SETQ L (CDR L))
          (GO LAB)))
EXPR)

(DEFPROP DSUBS
 (LAMBDA NIL
  (PROG (L)
        (ESUBS)
        (SETQ L (LIST (QUOTE UNIV)))
   LAB  (DSUB (CAR L))
        (SETQ L (APPEND L (GET (CAR L) (QUOTE OBJECTSP))))
        (SETQ L (CDR L))
        (COND ((NULL L) (RETURN T)) (T (GO LAB)))))
EXPR)

(DEFPROP DSUB
 (LAMBDA(NAME)
  (PROG (LST P)
```

```
        (SETQ INT (GET NAME (QUOTE INTENTIONP)))
        (COND ((NULL INT) (RETURN NIL)))
        (SELECT 2)
        (SETQ P (CAR INT))
        (SETPOS (CAR P) (CADR P))
    LAB (SETQ INT (CDR INT))
        (COND ((NULL INT) (RETURN T)))
        (LINETO (CAAR INT) (CADAR INT))
        (GO LAB)))
EXPR)

(DEFPROP ESUBS
 (LAMBDA NIL (PROG NIL (SELECT 2) (CLEAR)))
EXPR)

(DEFPROP CBPTS
 (LAMBDA(NAME1 NAME2)
  (PROG (B1 B2 S)
        (COMMENT READS
                 TWO
                 POINTS
                 MAKES
                 THEM
                 THE
                 BNDY
                 OF
                 BELONGS
                 SUB)
        (SETQ NAME1 (CAR (POINTER NAME1)))
        (SETQ NAME2 (CAR (POINTER NAME2)))
        (SETQ B1 (GET NAME1 (QUOTE BELONGSP)))
        (SETQ B2 (GET NAME2 (QUOTE BELONGSP)))
        (COND
         ((NEQ B1 B2)
          (RETURN
           (QUOTE
            (ERROR BNDY
                   POINTS
                   BELONG
                   TO
                   DIFFERENT
                   SUBPROBLEMS)))))
        (SETQ S (GET B1 (QUOTE STRUCTURELP)))
        (PUTPROP B1
                 (LIST (LIST NAME1 NAME2) (CADR S))
                 (QUOTE STRUCTURELP))
        (RETURN (LIST NAME1 NAME2))))
EXPR)

(DEFPROP DISTANCE1
 (LAMBDA(P1 P2)
  (SQRT
   (*PLUS (SQ (*DIF (CAR P1) (CAR P2)))
          (SQ (*DIF (CADR P1) (CADR P2))))))
EXPR)

(DEFPROP ***POINTER
 (LAMBDA NIL
  (PROG (P L BNDY MIN MIP MINT X)
        (COMMENT DETERMINES WHICH MINT IS BEING POINTED)
```

```
        (COMMENT AT
                 AND
                 PRINTS
                 A
                 (QUOTE P)
                 AT
                 THE
                 CLOSEST
                 VERTEX)
        (SETQ P (READ))
        (SETQ L
              (GET (GET (QUOTE STATUS) (QUOTE CONTEXT))
                   (QUOTE OBJECTSP)))
        (SETQ MIN 1222221)
        (SETQ MINP NIL)
        (SETQ MINNT NIL)
   LAB1 (SETQ BNDY (GET (CAR L) (QUOTE INTENTIONP)))
   LAB2 (COND ((NULL BNDY)
               (SETQ L (CDR L))
               (COND ((NULL L)
                      (SELECT 7)
                      (CLEAR)
                      (TEXT (QUOTE P)
                            (CAR MINP)
                            (CADR MINP))
                      (RETURN (LIST MINNT MINP)))
                     (T (GO LAB1))))
              (T (SETQ X (DISTANCE P (CAR BNDY)))
                 (COND
                  ((LESSP X MIN) (SETQ MIN X)
                                 (SETQ MINNT (CAR L))
                                 (SETQ MINP (CAR BNDY))))
                 (SETQ BNDY (CDR BNDY))
                 (GO LAB2)))))
EXPR)

(DEFPROP **POINTER
 (LAMBDA NIL
  (PROG (P L BNDY MIN MINP MINNT X)
        (COMMENT DETERMINES WHICH NINT IS BEING POINTED)
        (COMMENT AT
                 AND
                 PRINTS
                 A
                 (QUOTE P)
                 AT
                 THE
                 CLOSEST
                 VERTEX)
        (SETQ P (READ))
        (SETQ L (LIST (GET (QUOTE STATUS) (QUOTE CONTEXT))))
        (SETQ MIN 1222221)
        (SETQ MINP NIL)
        (SETQ MINNT NIL)
   LAB1 (SETQ BNDY (GET (CAR L) (QUOTE INTENTIONP)))
   LAB2 (COND ((NULL BNDY)
               (SETQ L
                     (APPEND
                      L
                      (GET (CAR L) (QUOTE OBJECTSP))))
```

```
                (SETQ L (CDR L))
                (COND ((NULL L)
                       (SELECT 7)
                       (CLEAR)
                       (TEXT (QUOTE P)
                             (CAR MINP)
                             (CADR MINP))
                       (RETURN (LIST MINNT MINP)))
                      (T (GO LAB1))))
               (T (SETQ X (DISTANCE P (CAR BNDY)))
                  (COND
                   ((LESSP X MIN) (SETQ MIN X)
                                  (SETQ MINNT (CAR L))
                                  (SETQ MINP (CAR BNDY))))
                  (SETQ BNDY (CDR BNDY))
                  (GO LAB2)))))
EXPR)

(DEFPROP *POINTER
 (LAMBDA NIL
  (PROG (P L BNDY MIN MINP MINNT X)
        (COMMENT DETERMINES WHICH NINT IS BEING POINTED)
        (COMMENT AT
                 AND
                 PRINTS
                 A
                 (QUOTE P)
                 AT
                 THE
                 CLOSEST
                 VERTEX)
        (SETQ P (READ))
        (SETQ L (LIST (QUOTE UNIV)))
        (SETQ MIN 1222221)
        (SETQ MINP NIL)
        (SETQ MINNT NIL)
   LAB1 (SETQ BNDY (GET (CAR L) (QUOTE INTENTIONP)))
   LAB2 (COND ((NULL BNDY)
               (SETQ L
                     (APPEND
                      L
                      (GET (CAR L) (QUOTE OBJECTSP))))
               (SETQ L (CDR L))
               (COND ((NULL L)
                      (SELECT 7)
                      (CLEAR)
                      (TEXT (QUOTE P)
                            (CAR MINP)
                            (CADR MINP))
                      (RETURN (LIST MINNT MINP)))
                     (T (GO LAB1))))
              (T (SETQ X (DISTANCE P (CAR BNDY)))
                 (COND
                  ((LESSP X MIN) (SETQ MIN X)
                                 (SETQ MINNT (CAR L))
                                 (SETQ MINP (CAR BNDY))))
                 (SETQ BNDY (CDR BNDY))
                 (GO LAB2)))))
 EXPR)
```

```
(DEFPROP ESYNTHS
 (LAMBDA NIL (PROG NIL (SELECT 6) (CLEAR)))
EXPR)

(DEFPROP DSYNTH
 (LAMBDA(NAME)
  (PROG (L P)
        (COMMENT DISPLAYSCSYNTHESIZED SOLUTION)
        (SELECT 6)
        (SETQ NAME (CAR (POINTER NAME)))
        (SETQ L (GET NAME (QUOTE SOLUTIONP)))
        (COND ((NULL L) (RETURN NIL)))
        (SETQ P (GET (CAR L) (QUOTE CENTROIDP)))
        (SETPOS (CAR P) (CADR P))
   LAB  (SETQ L (CDR L))
        (COND ((NULL L) (RETURN T)))
        (SETQ P (GET (CAR L) (QUOTE CENTROIDP)))
        (LINETO (CAR P) (CADR P))
        (GO LAB)))
EXPR)

(DEFPROP DINPUT
 (LAMBDA (NAME) (SETQ CITIES (DSKIN NAME)))
EXPR)

(DEFPROP RINPUT
 (LAMBDA NIL
  (PROG (P)
        (CLEARALL)
        (SELECT 1)
        (CLEAR)
        (SETQ CITIES NIL)
   LAB  (SETQ P (READ))
        (POINT (CAR P) (CADR P))
        (COND
         ((AND (ZEROP (CAR P)) (ZEROP (CADR P)))
          (RETURN CITIES)))
        (SETQ CITIES (CONS P CITIES))
        (GO LAB)))
EXPR)

(DEFPROP KSUB
 (LAMBDA(NAME)
  (PROG (B)
        (COMMENT DELETES
                 A
                 SUBPROBLEM
                 WILL
                 NOT
                 DELETE
                 UNIV
                 DELETES)
        (COMMENT STRUCTURE IN BELONGS RESTORES ITS)
        (COMMENT OBJECTS
                 TO
                 BELONGS
                 SUBPROB
                 ALLOWS
                 CITY
                 DELETION)
```

```
        (COND ((EQ NAME (QUOTE UNIV)) (RETURN NIL)))
        (SETQ NAME (CAR (POINTER NAME)))
        (SETQ B (GET NAME (QUOTE BELONGSP)))
        (PUTPROP B NIL (QUOTE SOLUTIONP))
        (PUTPROP B
                 (LIST (LIST NIL NIL) NIL)
                 (QUOTE STRUCTUREP))
        (PUTPROP
         B
         (APPEND (GET NAME (QUOTE OBJECTSP))
                 (REMOVEX NAME (GET B (QUOTE OBJECTSP))))
         (QUOTE OBJECTSP))
        (ESYNTHS)
        (ESOLS)
        (ESUBS)
        (OSUBS)
        (RETURN NAME)))
EXPR)

(DEFPROP COST
 (LAMBDA(NAME MODE)
  (PROG (L COUNT)
        (COMMENT COMPUTES
                 THE
                 COST
                 OF
                 A
                 SUBSOL
                 OR
                 ACSYNTHSOL)
        (COMMENT DEPENDING ON MODE)
        (SETQ NAME (CAR (POINTER NAME)))
        (COND ((EQ MODE (QUOTE SYNTH))
               (SETQ L (GET NAME (QUOTE SOLUTIONP))))
              (T
               (SETQ L
                     (CADR (GET NAME (QUOTE STRUCTUREP))))))
        (COND ((NULL L) (RETURN NIL)))
        (SETQ COUNT 0)
   LAB  (COND ((NULL (CDR L)) (RETURN COUNT)))
        (SETQ COUNT
              (*PLUS COUNT
                     (DISTANCE1
                      (GET (CAR L) (QUOTE CENTROIDP))
                      (GET (CADR L) (QUOTE CENTROIDP)))))
        (SETQ L (CDR L))
        (GO LAB)))
EXPR)

(DEFPROP KSYNTH
 (LAMBDA(NAME)
  (PROG NIL
        (COMMENT DELETES ACSYNTHESIZED SOLUTION)
        (SETQ NAME (CAR (POINTER NAME)))
        (PUTPROP NAME NIL (QUOTE SOLUTIONP))
        (ESYNTHS)))
EXPR)

(DEFPROP KSOL
 (LAMBDA(NAME)
```

```
 (PROG (L)
       (COMMENT DELETES A SUBPROBLEM SOLUTION)
       (SETQ NAME (CAR (POINTER NAME)))
       (PUTPROP
        NAME
        (LIST (CAR (GET NAME (QUOTE STRUCTUREP))) NIL)
        (QUOTE STRUCTUREP))
       (ESOLS)
       (DSOLS)))
EXPR)

(DEFPROP SYNTH1
 (LAMBDA(E)
  (PROG NIL
       (SETQ S (CADR (GET E (QUOTE STRUCTUREP))))
       (COND ((NULL S) (SETQ G (CONS E G)))
             (T (MAPCAR (QUOTE SYNTH1) S)))
       (RETURN NIL)))
EXPR)

(DEFPROP CSYNTH
 (LAMBDA(NAME)
  (PROG (G)
       (COMMENT SYNTHESIZES
                SOLUTION
                IN
                TREE
                ROOTED
                AT
                NAME)
       (SETQ NAME (CAR (POINTER NAME)))
       (SYNTH2 NAME)
       (COND ((EQ NAME (QUOTE UNIV))
              (MAPCAR
               (FUNCTION SYNTH1)
               (CONS (FUDGE NAME)
                     (CDADR
                      (GET NAME (QUOTE STRUCTUREP))))))
             (T (SYNTH1 NAME)))
       (PUTPROP NAME G (QUOTE SOLUTIONP))
       (ESOLS)
       (ESYNTHS)
       (DSYNTH NAME)
       (DSOLSEX NAME)
       (RETURN (COST NAME (QUOTE SYNTH))))))
EXPR)

(DEFPROP ROUND
 (LAMBDA (X) (FIX (*PLUS X 0.5)))
EXPR)

(DEFPROP BETWEENEQ
 (LAMBDA(X X1 X2)
  (OR (AND (LESSP X1 X) (LESSEQP X X2))
      (AND (LESSEQP X2 X) (LESSP X X1))))
EXPR)

(DEFPROP ISECT
 (LAMBDA(X X1 Y1 X2 Y2)
  (ROUND
```

```
   (*PLUS Y1
          (*TIMES (*DIF X X1)
                  (*QUO (*DIF Y2 Y1) (*DIF X2 X1))))))
EXPR)

(DEFPROP DISTANCE
 (LAMBDA(P1 P2)
  (*PLUS (SQ (*DIF (CAR P1) (CAR P2)))
         (SQ (*DIF (CADR P1) (CADR P2))))))
EXPR)

(DEFPROP ESOLS
 (LAMBDA NIL (PROG NIL (SELECT 5) (CLEAR) (RETURN NIL)))
EXPR)

(DEFPROP REMOVEX
 (LAMBDA(E L)
  (COND ((NULL L) NIL)
        ((NEQ E (CAR L)) (CONS (CAR L) (REMOVEX E (CDR L))))
        (T (CDR L))))
EXPR)

(DEFPROP DSOLS
 (LAMBDA NIL
  (PROG (L)
        (COMMENT DISPLAYS ALL SUBPROBLEM SOLUTIONS)
        (SELECT 5)
        (CLEAR)
        (SETQ L (LIST (QUOTE UNIV)))
   LAB  (SETQ L (APPEND L (GET (CAR L) (QUOTE OBJECTSP))))
        (DSOL (CAR L))
        (SETQ L (CDR L))
        (COND ((NULL L) (RETURN NIL)) (T (GO LAB))))))
EXPR)

(DEFPROP DSOL
 (LAMBDA(NAME)
  (PROG (SOL P)
        (COMMENT DISPLAYS SOLUTION TO SUBPROBLEM NAME)
        (SETQ NAME (CAR (POINTER NAME)))
        (SELECT 5)
        (SETQ SOL (CADR (GET NAME (QUOTE STRUCTUREP))))
        (COND ((NULL SOL) (RETURN NIL)))
        (SETQ P (GET (CAR SOL) (QUOTE CENTROIDP)))
        (SETPOS (CAR P) (CADR P))
   LAB  (SETQ SOL (CDR SOL))
        (COND ((NULL SOL) (RETURN T)))
        (SETQ P (GET (CAR SOL) (QUOTE CENTROIDP)))
        (LINETO (CAR P) (CADR P))
        (GO LAB)))
EXPR)

(DEFPROP EXIT
 (LAMBDA(NAME)
  (PROG (S)
        (COMMENT SETS UP OBJECT AS EXIT FROM SUBPROBLEM IT)
        (COMMENT BELONGS TO)
        (SETQ NAME (CAR (POINTER NAME)))
        (SETQ S
              (GET (GET NAME (QUOTE BELONGSP))
```

```
                    (QUOTE STRUCTUREP)))
          (COND
           ((NOT (NULL (CADAR S)))
            (PRINT (QUOTE (CHANGE EXIT POINT?)))
            (COND ((EQUAL (READ) (QUOTE NO)) (RETURN NIL)))))
          (PUTPROP (GET NAME (QUOTE BELONGSP))
                   (LIST (LIST (CAAR S) NAME) (CADR S))
                   (QUOTE STRUCTUREP))
          (RETURN NAME)))
EXPR)

(DEFPROP ENTRY
 (LAMBDA(NAME)
  (PROG (S)
        (COMMENT SETS
                 AN
                 OBJECT
                 AS
                 ENTRY
                 TO
                 SUBPROBLEM
                 IT
                 BELONGS
                 TO)
        (SETQ NAME (CAR (POINTER NAME)))
        (SETQ S
              (GET (GET NAME (QUOTE BELONGSP))
                   (QUOTE STRUCTUREP)))
        (COND
         ((NOT (NULL (CAAR S)))
          (PRINT (QUOTE (CHANGE ENTRY POINT?)))
          (COND ((EQUAL (READ) (QUOTE NO)) (RETURN NIL)))))
        (PUTPROP (GET NAME (QUOTE BELONGSP))
                 (LIST (LIST NAME (CADAR S)) (CADR S))
                 (QUOTE STRUCTUREP))
        (RETURN NAME)))
EXPR)

(DEFPROP POINTER
 (LAMBDA(NAME)
  (COND ((EQ NAME (QUOTE *)) (*POINTER))
        ((EQ NAME (QUOTE **)) (**POINTER))
        ((EQ NAME (QUOTE ***)) (***POINTER))
        (T
         (COND ((NOT (NULL (GET NAME (QUOTE INTENTIONP))))
                (LIST NAME
                      (CAR (GET NAME (QUOTE INTENTIONP)))))
               (T (LIST NAME NIL)))))))
EXPR)

(DEFPROP CONTEXT
 (LAMBDA(NAME)
  (PROG (X)
        (SETQ X (POINTER NAME))
        (SETQ NAME (CAR X))
        (PUTPROP (QUOTE STATUS) NAME (QUOTE CONTEXT))
        (SELECT 7)
        (CLEAR)
        (SELECT 4)
        (CLEAR)
```

```
        (COND
         ((NOT (NULL (CADR X)))
          (TEXT (QUOTE C) (CAADR X) (CADADR X))))
        (RETURN NAME)))
EXPR)

(DEFPROP CSUB
 (LAMBDA(NAME)
  (PROG (CON FP BP OBJ NOBJ BNDY %DUM)
        (COMMENT READS BOUNDARY CONSTRUCTS LIST OF ENCLOSED)
        (COMMENT OBJECTS
                 FROM
                 PRESENT
                 CONTEXT
                 REMOVEXS
                 OBJECTS
                 FROMM)
        (COMMENT CONTEXT MAKES THEM OBJECTS OF NEW NONT)
        (COMMENT ADDS NEW NONT TO CONTEXT OBJECTS)
        (SETQ BNDY (READBNDY))
        (COND ((NULL BNDY) (RETURN NIL)))
        (COND
         ((EQ NAME (QUOTE *))
          (SETQ NAME
                (READLIST
                 (APPEND (QUOTE (S P)) (EXPLODE NTNUM))))
          (SETQ NTNUM (ADD1 NTNUM))))
        (SETQ CON (GET (QUOTE STATUS) (QUOTE CONTEXT)))
        (SETQ OBJ (GET CON (QUOTE OBJECTSP)))
        (SETQ NOBJ NIL)
        (SETQ BP OBJ)
        (SETQ OBJ (CONS (QUOTE %DUM) OBJ))
        (SETQ FP OBJ)
        (COMMENT CHECK THE CONTEXT OBJECT LIST)
  LAB1  (COND
         ((NOT (NULL BP))
          (COND ((INTERIOR (GET (CAR BP) (QUOTE CENTROIDP))
                           BNDY)
                 (SETQ NOBJ (CONS (CAR BP) NOBJ))
                 (PUTPROP (CAR BP) NAME (QUOTE BELONGSP))
                 (RPLACD FP (CDR BP))
                 (SETQ BP (CDR FP)))
                (T (SETQ FP BP) (SETQ BP (CDR BP))))
          (GO LAB1)))
        (COMMENT SETUP PROPERTIES OF NEW NONTS)
        (PUTPROP NAME
                 (LIST (LIST NIL NIL) NIL)
                 (QUOTE STRUCTURFP))
        (PUTPROP NAME NOBJ (QUOTE OBJECTSP))
        (PUTPROP NAME BNDY (QUOTE INTERIORP))
        (PUTPROP NAME CON (QUOTE BELONGSP))
        (PUTPROP NAME (CENTROIDF NOBJ) (QUOTE CENTROIDP))
        (COMMENT FIXUP CONTEXT NONT)
        (NCONC OBJ (LIST NAME))
        (PUTPROP CON (CDR OBJ) (QUOTE OBJECTSP))
        (PUTPROF CON
                 (LIST (LIST NIL NIL) NIL)
                 (QUOTE STRUCTURFP))
        (PUTPROF CON NIL (QUOTE SOLUTIONP))
        (PUTPROF CON
```

```
                (CENTROIDF (CDR OBJ))
                (QUOTE CENTROIDP))
        (RETURN NAME)))
EXPR)

(DEFPROP CENTROIDF
 (LAMBDA(L)
  (PROG (P N X Y)
        (COMMENT COMPUTES CENTROID OF CENTROIDS OF NTS IN L)
        (COND ((NULL L) (RETURN NIL)))
        (SETQ N (LENGTH L))
        (SETQ X 0)
        (SETQ Y 0)
   LAB  (COND
         ((NULL L)
          (COND ((ZEROP X) (RETURN NIL))
                (T (RETURN (LIST (*QUO X N) (*QUO Y N)))))))
        (SETQ P (GET (CAR L) (QUOTE CENTROIDP)))
        (COND
         ((NOT (NULL P)) (SETQ X (*PLUS X (CAR P)))
                         (SETQ Y (*PLUS Y (CADR P)))))
        (SETQ L (CDR L))
        (GO LAB)))
EXPR)

(DEFPROP EVENP
 (LAMBDA (N) (ZEROP (REMAINDER N 2)))
EXPR)

(DEFPROP LESSEQP
 (LAMBDA (X Y) (OR (EQUAL X Y) (LESSP X Y)))
EXPR)

(DEFPROP BETWEEN
 (LAMBDA(X X1 X2)
  (OR (AND (LESSEQP X1 X) (LESSEQP X X2))
      (AND (LESSEQP X2 X) (LESSEQP X X1))))
EXPR)

(DEFPROP INTERIOR
 (LAMBDA(P L)
  (PROG (X Y X1 Y1 X2 Y2 NX1 NX2 NY1 NY2 L1 SEG COUNT Z)
        (COMMENT DETERMINES
                 IF
                 PLIES
                 ON
                 OR
                 INSIDE
                 THE
                 POLYGON)
        (COMMENT DESCRIBED BY L RETURNS T OR NIL WILL WORK)
        (COMMENT FOR NOT CONVEX POLYGONS)
        (SETQ L1 L)
        (SETQ X (CAR P))
        (SETQ Y (CADR P))
        (SETQ COUNT 0)
   LAB1 (SETQ X1 (*PLUS 0.0 (CAAR L1)))
        (SETQ Y1 (CADAR L1))
        (SETQ X2 (CAADR L1))
        (SETQ Y2 (CADADR L1))
```

```
        (SETQ L1 (CDR L1))
        (COND ((NOT (EQUAL X1 X2)) (GO LAB3)))
   LAB2 (COND ((NULL (CDR L1))
               (COND ((EVENP COUNT) (RETURN NIL))
                     (T (RETURN T))))
              (T (GO LAB1)))
   LAB3 (COND ((NOT (BETWEENEQ X X1 X2)) (GO LAB2)))
        (COND ((NOT (EQUAL X X2)) (GO LAB5)))
        (COND ((EQUAL Y Y2) (RETURN T)))
   LAB4 (COND ((NULL (CDR L1))
               (SETQ NX1 (*PLUS 0.0 (CAAR L)))
               (SETQ NY1 (CADAR L))
               (SETQ NX2 (CAADR L))
               (SETQ NY2 (CADADR L))
               (SETQ L (CDR L)))
              (T (SETQ NX1 (*PLUS 0.0 (CAAR L1)))
                 (SETQ NY1 (CADAR L1))
                 (SETQ NX2 (CAADR L1))
                 (SETQ NY2 (CADADR L1))
                 (SETQ L1 (CDR L1))))
        (COND ((EQUAL NX1 NX2)
               (COND ((BETWEEN Y NY1 NY2) (RETURN T))
                     (T (GO LAB4))))
              (T (COND ((GREATERP Y NY1) (GO LAB2)))
                 (SETQ X (ADD1 X))
                 (COND
                  ((BETWEEN X X1 X2)
                   (SETQ Z (ISECT X X1 Y1 X2 Y2))
                   (COND
                    ((LESSEQP Y Z)
                     (SETQ COUNT (ADD1 COUNT)))))))
                 (COND
                  ((BETWEEN X X2 NX2)
                   (SETQ Z (ISECT X X2 Y2 NX2 NY2))
                   (COND
                    ((LESSEQP Y Z)
                     (SETQ COUNT (ADD1 COUNT)))))))
                 (SETQ X (SUB1 X))
                 (GO LAB2)))
   LAB5 (SETQ Z (ISECT X X1 Y1 X2 Y2))
        (COND ((EQUAL Y Z) (RETURN T)))
        (COND ((LESSP Y Z) (SETQ COUNT (ADD1 COUNT))))
        (GO LAB2)))
EXPR)

(DEFPROP UTIL
 (UTIL FUDGE
       ECITIES
       OSYNTPS
       ECITYNAMES
       OCITYNAMES
       ESUBNAMES
       EGARBAGE
       CEPTS1
       OSUBSEX
       OSUBNAMES
       OSUBNAME
       OCITIES
       ONAMES
       OUT
```

```
        SLAST
        IN
        SYNTH2
        DSUBS
        DSUB
        ESUBS
        CPPTS
        DISTANCE1
        ***POINTER
        **POINTER
        *POINTER
        ESYNTHS
        DSYNTH
        DINPUT
        RINPUT
        KSUB
        COST
        KSYNTH
        KSOL
        SYNTH1
        CSYNTH
        ROUND
        BETWEENEQ
        ISECT
        DISTANCE
        ESOLS
        REMOVEX
        DSOLS
        DSOL
        EXIT
        ENTRY
        POINTER
        CONTEXT
        CSUB
        CENTROIDF
        EVENP
        LESSEQP
        BETWEEN
        INTERIOR
        UTIL
        READBNDY)
VALUE)

(DEFPROP READBNDY
 (LAMBDA NIL
  (PROG (L X0 Y0 P X Y)
        (COMMENT READS
                 LIST
                 OF
                 BNDY
                 VERTICES
                 FROM
                 PCN
                 DISPLAYS)
        (COMMENT BNDY LIST ENDS WITH (A A) IF BNDYLIST LESS)
        (COMMENT THAN
                 THREE
                 POINTS
                 ERASES
                 PARTIAL
```

```
                RETURNS
                NIL)
        (SELECT 2)
        (SETQ P (READ))
        (SETQ X0 (CAR P))
        (SETQ Y0 (CADR P))
        (COND ((AND (ZEROP X0) (ZEROP Y0)) (RETURN NIL)))
        (POINT X0 Y0)
        (SETQ L (LIST (LIST X0 Y0)))
LAB     (SETQ P (READ))
        (SETQ X (CAR P))
        (SETQ Y (CADR P))
        (COND
         ((AND (ZEROP X) (ZEROP Y))
          (COND ((LESSP (LENGTH L) 3)
                 (CLEAR)
                 (DISPRNCY)
                 (RETURN NIL))
                (T (LINETO X0 Y0)
                   (RETURN (CONS (LIST X0 Y0) L))))))
        (LINETO X Y)
        (SETQ L (CONS (LIST X Y) L))
        (GO LAB)))
EXPR)
```

References

1. Howden, William E., Plans and Problem Solving Structure, U.C.I. Tech Report, (in preparation).

2. Newman, William, Input and Output Functions of IMSYS, U.C.I. Graphics Memo #3, June, 1971.

3. Bobrow, Robert, et al, U.C.I. New-Lisp Manual, University of California at Irvine.

4. Held, Michael and Karp, Richard M., A Dynamic Programming Approach to Sequencing Problems, Journal for the Society of Industrial and Applied Mathematics, Vol. 10, No. 1, March, 1962.

5. Bellmore, M. and Nemhauser, G. L., The Travelling Salesman Problem: A Survey, Operations Research, Vol. 16, 1968.

6. Lin, Shen, Computer Solutions of the Travelling Salesman Problem, The Bell System Technical Journal, December, 1965.

7. Croes, G. A., A Method for Solving Travelling Salesman Problems, Operations Research, Vol. 5.

8. LIFE World Library, Atlas of the World, TIME Incorporated, New York, 1966.