An Effective Heuristic Algorithm for the Traveling-Salesman Problem

S. Lin and B. W. Kernighan

Bell Telephone Laboratories, Incorporated, Murray Hill, N.J.

(Received October 15, 1971)

This paper discusses a highly effective heuristic procedure for generating optimum and near-optimum solutions for the symmetric traveling-salesman problem. The procedure is based on a general approach to heuristics that is believed to have wide applicability in combinatorial optimization problems. The procedure produces optimum solutions for all problems tested, 'classical' problems appearing in the literature, as well as randomly generated test problems, up to 110 cities. Run times grow approximately as n^2; in absolute terms, a typical 100-city problem requires less than 25 seconds for one case (GE635), and about three minutes to obtain the optimum with above 95 per cent confidence.

THE SYMMETRIC traveling-salesman problem is: Given an n by n symmetric matrix of distances between n cities, find a minimum-length tour that visits each city exactly once. Of course, instead of distance, other notions such as time, cost, etc., can be considered; we will use 'distance' to represent any such measure. This problem has been studied for many years, with limited success.[1] Exact algorithms may require inordinate running times; heuristic methods produce good answers for somewhat larger problems in reasonable times, but provide no guarantee that the optimum answer will appear. And with current heuristics, effectiveness diminishes and run time increases rapidly with n, so there has been little work done on problems larger than about 60 cities.

For instance, recent work by HELD AND KARP[5] gives a method that solves exactly a certain class of traveling-salesman problems in a reasonable time. However if the particular problem turns out not to be in this class, the procedure must be supplemented by an added mechanism—in this case, branch and bound—and the run time is often prohibitive. The largest problem they report on is 64 cities.

A quite different approach is taken by KROLAK ET AL.[9] who use several fast, simple, and less effective heuristics to achieve good solutions, and then apply human judgment to pictures of the tour ("man-machine interaction") to try for optimum solutions. While this approach has been applied to large problems (200 cities), it is costly in machine and especially man time, and the results are generally suboptimal. (We have improved on three of their five 100-city problems.) Furthermore, pictorial methods break down for problems that are not 2-dimensional and Euclidean.

This paper discusses a new heuristic method that produces optimum solutions with high frequency, in running times that grow about as n^2. The procedure is based on a general approach to heuristics that we believe to be of wide applicability. It has already been applied with considerable success to graph partitioning.[8] (The graph partitioning problem is: given a graph on the set of nodes S, find a

subset of the nodes T that is minimally connected to S-T and satisfies the constraint that T and S-T have the same number of nodes.) In this paper we concentrate on its application to the traveling-salesman problem. We will discuss the basic philosophy involved in the heuristic procedure, describe our procedure in its terms, and illustrate its effectiveness with computational experience on a variety of traveling-salesman problems.

Many combinatorial optimization problems like graph partitioning and the traveling-salesman problem can be formulated in the abstract as 'find from a set S a subset T that satisfies some criterion C and minimizes an objective function f.' For example, in the traveling-salesman problem, we have to find, from the set of all edges of a complete graph, a subset that forms a tour and has minimum length.

For most of these problems, all known algorithms require computing times that grow exponentially with n. (Recent work in complexity theory[7] indicates that problems like the traveling-salesman problem very probably are inherently exponential.) Heuristic methods appear to be the only feasible line of attack. From a theoretical standpoint, although we cannot generally prove optimality of solutions, we can obtain statistical confidence; for practical applications, frequently all that matters is that good answers are obtained in feasible running times.

One basic approach to heuristics for combinatorial optimization problems is iterative improvement of a set of randomly selected feasible solutions:

1. Generate a pseudorandom feasible solution, that is, a set T that satisfies C.

2. Attempt to find an improved feasible solution T' by some transformation of T.

3. If an improved solution is found, i.e., $f(T') < f(T)$, then replace T by T' and repeat from Step 2.

4. If no improved solution can be found, T is a locally optimum solution. Repeat from Step 1 until computation time runs out, or the answers are satisfactory.

The actual heuristic procedure (the transformation of Step 2) maps the random starting solutions of Step 1 into locally optimum solutions, among which the global optimum will hopefully appear. The better the heuristic is, the smaller the set of local optima will be, and the higher will be the fraction of random starts that lead to the global optimum. Random, uniformly distributed starting solutions are chosen in Step 1 (rather than, say, good solutions), unless we know in advance that a particular kind of starting solution leads to better answers. There are two reasons for this. First, a worthwhile heuristic should produce 'good' starting solutions just as fast as any other starting procedure—this is certainly our experience. Second, constructive solutions are usually deterministic, so that it may not be possible to get more than one initial solution.

The heart of the iterative procedure is, of course, Step 2, the process that tries to improve upon a given solution. One transformation that has been applied to a variety of problems[2,10–12] is the exchange of a fixed number k of elements from T with k elements from S-T, such that the resulting solution T'' is feasible and better. This is repeated as long as such groups can be found. Eventually it will not be possible to improve T further by such exchanges, at which time we have a locally optimum solution. Naturally enough, the whole problem is finding the right elements to exchange, for one can always find optimum solutions by exchanging the correct groups.

This interchange strategy was applied to the traveling-salesman problem by CROES,[2] with k fixed at 2, and by LIN,[11] with $k=3$, with considerable success.

But having to specify the value of k in advance is a serious drawback. Computational effort rises rapidly with increasing k, and it is difficult to know in advance what k to use as the best compromise between running time and quality of solution.

Our heuristic method is based on a substantial generalization of the interchange transformation. Suppose T is a nonoptimal but feasible solution. Then we can say that T is not optimal because there are k elements $x_1, \cdots, x_k$ in T that are 'out of place'; to make T optimal, they should be replaced by k elements $y_1, \cdots, y_k$ of S-T. The problem is simply to identify k and the x's and y's.

Since we do not know what k should be, it seems artificial to fix k and then consider all possible k-subsets of T for this fixed k. Instead, we try to find k and $x_1, \cdots, x_k$ and $y_1, \cdots, y_k$ as best we can, *element by element.* Thus we will try first to identify x_1 and y_1, as the 'most-out-of-place' pair; then, with x_1 and y_1 selected and temporarily set aside, we find x_2 and y_2, the most-out-of-place pair in the remaining sets; and so on. More formally:

1. Generate a random initial solution T.

2. (a) Set $i=1$.

(b) Select x_i and y_i as the most-out-of-place pair at the ith step. This generally means that x_i and y_i are chosen to maximize the improvement when $x_1, \cdots, x_i$ are exchanged with $y_1, \cdots, y_i$. x_i is chosen from $T-\{x_1, \cdots, x_{i-1}\}$ and y_i from $S-T-\{y_1, \cdots, y_{i-1}\}$.

(c) If it appears that no more gain can be made, according to an appropriate stopping rule, go to Step 3; otherwise, set $i=i+1$ and go back to Step 2(b).

3. If the best improvement is found for $i=k$, exchange $x_1, \cdots, x_k$ with $y_1, \cdots, y_k$, to give a new T, and go to Step 2; if no improvement is found, go to Step 4.

4. Repeat from Step 1 if desired.

To make this work, clearly we need several things:

1. We need a selection rule that tells us quickly and effectively which pair is currently most out of place: quickly, for obvious reasons, and effectively, for a mistake at any stage may well destroy all hope of a good result from later stages.

2. We need a simple function that represents the total profit from a proposed set of exchanges. Suppose that g_i is the profit or 'gain' associated with the exchange of x_i and y_i, given that $x_1, y_1, \cdots, x_{i-1}, y_{i-1}$ have already been chosen. Then it is most useful if the gain from the exchange of $x_1, \cdots, x_i$ with $y_1, \cdots, y_i$ is $g_1+\cdots+g_i$, i.e., the gains are additive.

Given this additivity, there is no need for the selection process to cease immediately when some g_i is negative; in fact, we need stop only when $\sum_{i=1}^{i=k} g_i \leqq 0$ for all k of interest. This helps in evading local minima.

3. If we are to be able to stop this selection process for any value of k, and exchange $x_1, \cdots, x_k$ with $y_1, \cdots, y_k$, we must know that the exchange is *feasible*, that is, that the resulting solution satisfies the criterion C. Thus, we require that (for most k) each proposed exchange leave us in a feasible state. This saves us, first, from some serious bookkeeping headaches, and, second, from laboring hard to exchange sets, only to discover that they cannot profitably be made feasible after the exchange.

4. A stopping rule is needed to tell us that there can be no further profit in looking for elements to exchange, or at least that we have reached the point of diminishing returns. (This requires a delicate balance between optimism, which

can spend much time chasing fruitless paths, and pessimism, which misses golden opportunities.)

5. Last, we require that the sets $x_1, \cdots, x_k$ and $y_1, \cdots, y_k$ be disjoint. Once an element is moved one way, it is not returned during this iteration. (It may well move back upon the next, but that is another story.) This is largely pragmatic—it avoids various subtle implementation bugs, reduces running time, simplifies the gain function, and provides an effective stopping rule.

After we have examined some sequence $x_1, \cdots, x_m$ and $y_1, \cdots, y_m$ of proposed exchanges, with their corresponding gains $g_1, g_2, \cdots, g_m$, the actual value of k that defines the sets to exchange is of course the one for which $g_1+g_2+\cdots+g_k$ is maximum (and feasible). If this is positive, the sets are exchanged, to give a new and better solution T, and the process is iterated from this new starting point.

Eventually we reach a point where $g_1+\cdots+g_k$ is always zero or negative. This indicates that no further improvement can be made by the procedure, and so the solution at this time is our local optimum.

Although in the discussion above we have assumed that every feasible set has the same cardinality, the basic ideas can be readily adapted to transformations that do not preserve the sizes of T and S-T.

1. THE BASIC TRAVELING-SALESMAN ALGORITHM

To APPLY THIS method to the traveling-salesman problem, let S be the set of all links [i.e., $n(n-1)/2$ edges between the n cities], and let T be an n-subset of S that forms a tour (feasibility criterion C). We want to find a tour with minimum length (objective function f).

Consider an arbitrary tour T with length $f(T)$ and any tour T' with length $f(T')<f(T)$. Suppose T and T' differ (as sets of n links) by k links. Our basic algorithm attempts to transform T into T' by identifying *sequentially* the k pairs of links to be exchanged between T and S-T. That is, we attempt to find two sets of links $X=\{x_1, \cdots, x_k\}$ and $Y=\{y_1, \cdots, y_k\}$ such that, if the links in X are deleted or 'broken' and replaced by the links in Y, the result is a tour of lower cost.

Figure 1 illustrates the situation for $k=3$. Figure 1 (a) is the tour T, indicating X and Y; Fig. 1 (b) shows the resulting T'. Notice that we have numbered the affected links in a natural way: x_i and y_i share an endpoint, and so do y_i and x_{i+1}. ($x_{k+1}=x_1$). Generally it is possible to perform this numbering, and thus convert T to T' sequentially. (Figure 2 shows an example where this numbering is not possible. This situation arose only infrequently in problems we studied.)

Assuming the numbering can be performed, the sequence of exchanges we wish to find is $x_1, y_1; x_2, y_2; \cdots$, etc. Since we do not know what T' might be, the attempt is, of course, to find any sequence that reduces T to T' with $f(T')<f(T)$ and iterate the process on T' until no further reduction can be made.

Let the lengths of x_i and y_i be $|x_i|$ and $|y_i|$, respectively, and define $g_i=|x_i|-|y_i|$. This is the gain from exchanging x_i with y_i. Although some of the g_i may be negative, if $f(T')<f(T)$, clearly we have $\sum_1^k g_i=f(T)-f(T')>0$. Part of our procedure is based on the following simple fact: if a sequence of numbers has a positive sum, there is a cyclic permutation of these numbers such that *every partial sum* is

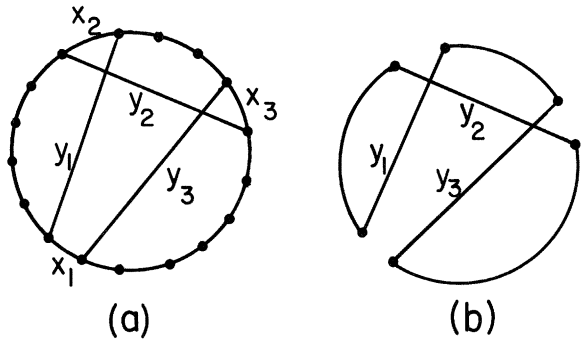

Fig. 1. Sequential exchange: (a) Tour T. (b) Tour T'.

positive. [*Proof.* Let k be the largest index for which $g_1+\cdots+g_{k-1}$ is minimum. If $k \leqq j \leqq n$,

$$g_k+\cdots+g_j=(g_1+\cdots+g_j)-(g_1+\cdots+g_{k-1})>0.$$

If $1 \leqq j<k$,

$$g_k+\cdots+g_n+g_1+\cdots+g_j \geqq g_k+\cdots+g_n+g_1+\cdots+g_{k-1}>0.]$$

In particular, then, since we are looking for sequences of g_i's that have a positive sum, *we need only consider sequences of gains whose partial sum is always positive.* This gain criterion enables us to reduce enormously the number of sequences we need to examine; it is the heart of our stopping rule.

We now present an outline of the basic heuristic algorithm. Figure 3 illustrates the notation. The description is annotated by a commentary in the legend; a simple example is worked out after the algorithm is described.

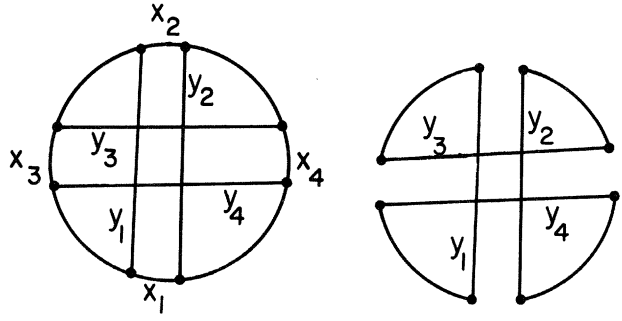

Fig. 2. Nonsequential exchange.

1. Generate a random starting tour T.

2. Set $G^*=0$. [G^* is the best improvement made so far.] Choose any node t_1 and let x_1 be one of the edges of T adjacent to t_1. Let $i=1$.

3. From the other endpoint t_2 of x_1, choose y_1 to t_3 with $g_1>0$. If no such y_1 exists, go to Step 6(d). [This is the first application of the gain criterion.]

4. Let $i=i+1$. Choose x_i [which currently joins t_{2i-1} to t_{2i}] and y_i as follows:

(a) x_i is chosen so that, if t_{2i} is joined to t_1, the resulting configuration is a tour. [Thus,

for a given y_{i-1}, x_i is uniquely determined. This is the application of the feasibility criterion; it *guarantees* that we can always 'close up' to a tour if we wish, simply by joining t_{2i} to t_1, for any $i \geqq 2$. The choice of y_{i-1}, Step 4(e), ensures that there is always such an x_i.]

(b) y_i is some available link at the endpoint t_{2i} shared with x_i, subject to (c), (d), and (e). If no y_i exists, go to Step 5. [Clearly, to make a large cost reduction at the ith step, $|y_i|$ should be small, and so in general we choose nearest neighbors preferentially. *See* Section 2B.]

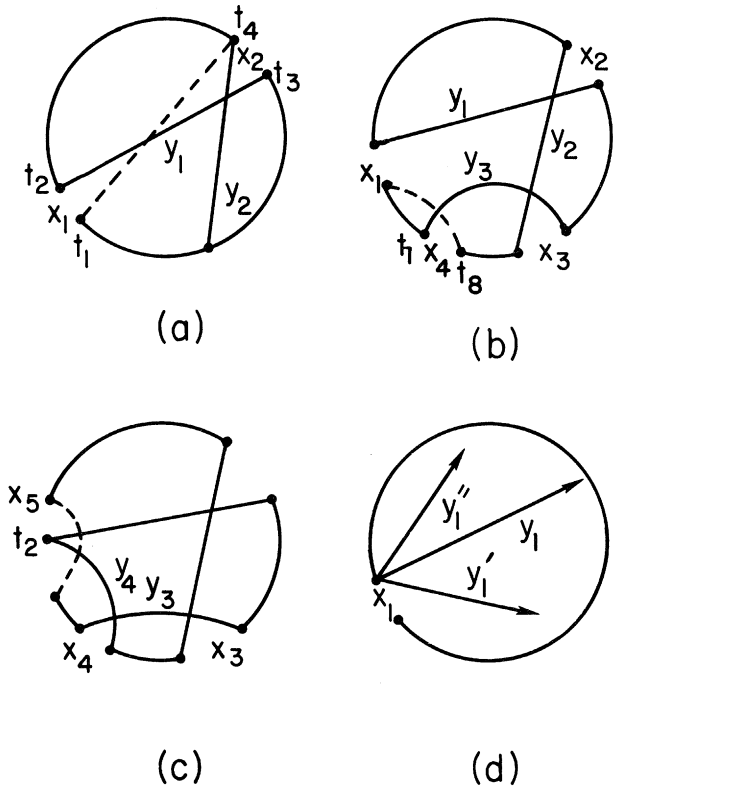

Fig. 3. Algorithm description, Step 4 (the dotted line indicates a possible close-up). (a) $i=2$. (b) $i=4$. Note that t_8 is uniquely determined. (c) $i=5$. Note both original edges at t_5 have been broken. (d) Backtracking in Step 6c: multiple choices for y_1.

(c) To guarantee that the x's and y's are disjoint, x_i cannot be a link previously joined (i.e., a $y_j, j<i$), and similarly y_i cannot be a link previously broken.

(d) $G_i = \sum_1^i g_j > 0$. [Gain criterion.]

(e) In order to ensure that the feasibility criterion of (a) can be satisfied at $i+1$, the y_i chosen must permit the breaking of an x_{i+1}.

(f) Before y_i is constructed, we check if closing up by joining t_{2i} to t_1 will give a gain value better than the best seen previously. [Since we have satisfied the feasibility criterion for $i \geqq 2$, we know this results in a tour.] Let y_i^* be a link connecting t_{2i} to t_1, and let $g_i^* = |y_i^*| - |x_i|$. If $G_{i-1}+g_i^* > G^*$, set $G^* = G_{i-1}+g_i^*$ and let $k=i$. [G^* is always the best im-

provement in T recorded so far, and is thus our standard of comparison. $G^* \geqq 0$, and is monotone nondecreasing. The index k defines the sets to be exchanged to achieve G^*.]

5. Terminate the construction of x_i and y_i in Steps 2 through 4 when either no further links x_i and y_i satisfy 4(c)–(e), or when $G_i \leqq G^*$. [This is our stopping criterion.] If $G^* > 0$, take the tour T' with $f(T') = f(T) - G^*$ and repeat the whole process from Step 2, using T' as the initial tour.

6. If $G^* = 0$, a limited backtracking facility is invoked, as follows:

(a) Repeat Steps 4 and 5, choosing y_2's in order of increasing length, as long as they satisfy the gain criterion $g_1 + g_2 > 0$. [If an improvement is found at any time, of course, this causes a return to Step 2.]

(b) If all choices of y_2 in Step 4(b) are exhausted without profit, return to Step 4(a) and try the alternate choice for x_2. [Breaking the alternate x_2 introduces a temporary violation of feasibility; we discuss this step in detail after the algorithm description.]

(c) If this also fails to give improvement, a further backup is performed to Step 3, where the y_1's are examined in order of increasing length.

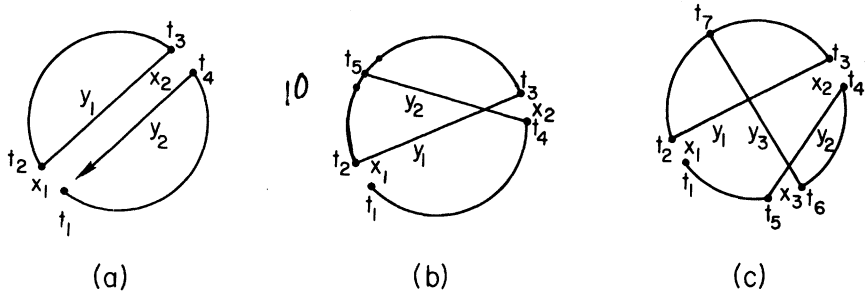

Fig. 4. The effect of alternate x_2. (a) Cannot close up at x_2. (b) Two choices for x_3. (c) Unique choice for x_3; limited choice for y_3.

(d) If the y_1's are also exhausted without profit, we try the alternate x_1 in Step 2.

(e) If this fails, a new t_1 is selected, and we repeat at Step 2.

[Note that backtracking is performed *only* if no gain can be found, and *only* at levels 1 and 2 ($i = 1$ and $i = 2$). We will discuss backtracking further in the next section.]

7. The procedure terminates when all n values of t_1 have been examined without profit. At this time, we may consider further random tours in Step 1.

This terminates the description of the algorithm, except for discussion of the effect of breaking an x_2 that lies between t_3 and t_1, as in Fig. 4(a). We allow this latter possibility because, although it adds some complexity, it substantially increases overall effectiveness. Choice of a nonfeasible alternate is allowed *only* for $i = 2$.

In Fig. 4(a), y_2 must not join to t_1, for this leaves two disconnected halves. Now, if t_5 is between t_2 and t_3, t_6 may be on either side of t_5, and, indeed, the second possibility will be investigated if the first does not lead to a profitable exchange. *See* Fig. 4(b).

If t_5 lies between t_1 and t_4, as in Fig. 4(c), there is only one choice for t_6 (it must be between t_5 and t_4, to satisfy the feasibility criterion) and t_7 must lie between t_2

and t_8. But then t_8 can be on either side of t_7; we choose the one for which $|x_4|$ is maximum.

In any case, from this point onward, we return to the normal course of the algorithm in Step 4.

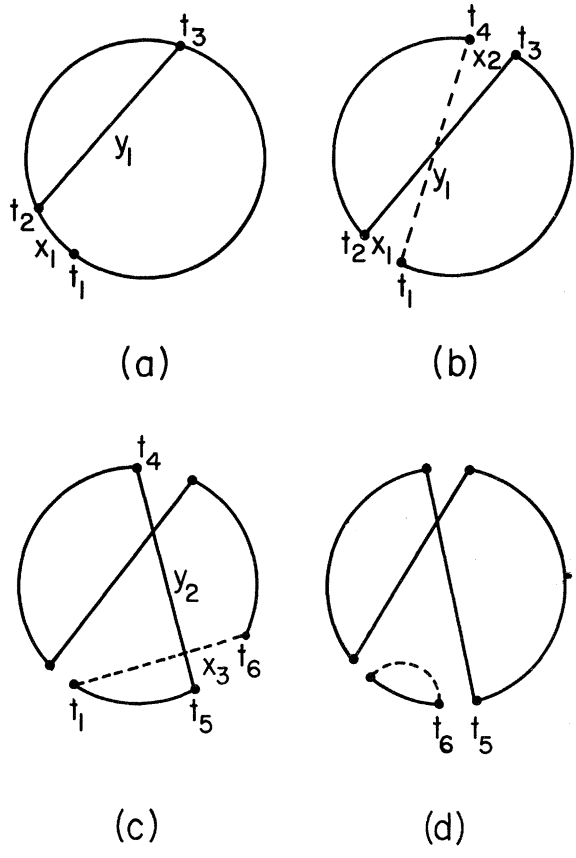

Fig. 5. An example. (a) Step 3. (b) Step 4; potential close-up for $i=2$. (c) Unique choice for x_3. Potential close-up for $i=3$. (d) Forbidden choice for x_3 and t_6.

An Example

Start with a tour T, which we show schematically in Fig. 5(a), and two adjacent nodes t_1 and t_2; x_1 is the link joining them. Let t_3 be the node nearest to t_2; y_1 is the edge (t_2, t_3). Since the x's and y's are to be disjoint, y_1 is not allowed to be either of the edges already connected to t_2. Set $g_1=|x_1|-|y_1|$; if this is not positive, we back up [Step 6(d)] and let t_2 be the other neighbor of t_1. Set $i=2$ and let t_4 be the neighbor of t_3, as shown in Fig. 5(b). x_2 is the edge (t_3, t_4).

Now *if* y_2 were chosen to join t_4 to t_1, the result would be a tour, and *if* $g_1+g_2>0$, we could improve T by exchanging x_1 and x_2 with y_1 and y_2. Remember this potential improve-

ment—this is G^*; $k=2$ in Step 4(f). Choose t_5 as the nearest neighbor of t_4, and let y_2 be the edge (t_4, t_5). Again t_5 cannot be either of the nodes already connected to t_4.

As we see in Fig. 5(c), there is only one choice for t_6 and x_3, if we wish to be able to close up immediately—if we let x_3 be the other edge connected to t_5, then the tour would become two separated pieces, as in Fig. 5(d).

Again, we check to see if immediate close-up (joining t_6 to t_1) gives a better gain than was available by joining t_4 to t_1, update G^* and set $k=3$. If $g_1+g_2+g_3 \leqq G^*$, the stopping rule says to take the profitable exchange at $k=2$. We are also done if the best choice for y_3 should happen to be (t_6, t_1). Otherwise, we continue by selecting t_7, and so on.

Computational experience shows that the algorithm does much less work than superficial examination might indicate, largely because the gain criterion is very strong. The computation time per local optimum depends on the amount of backtracking, but growth averages about $n^{2.2}$ as we have described it. We will discuss this further at the end of Section 3.

The feasibility criterion is satisfied by maintaining the configuration as a tour for $i \geqq 2$ or 3 (depending on which x_2 is chosen). Since at each stage we try to maximize G_k by choosing a small y_k, we are in effect choosing out-of-place elements. And by terminating as soon as $G_k \leqq G^*$, we reduce the likelihood of following long fruitless paths. The average amount of overshoot (i.e., depth of search minus value of k) is not large. Typically, the first several improvements found by the procedure have large k values: changes of $(\frac{3}{4})n$ with zero overshoot are common. Gradually improvement becomes harder to find, and the process settles down to a series of smaller changes (around 2–7), with some overshoot. The number of improvements is usually between $n/4$ and $n/3$.

It can be easily seen that the local optimum solutions obtained from this heuristic algorithm are necessarily 3-opt in the sense of reference 11, so the results are guaranteed as good as those obtained in reference 11, in substantially less time.

Backtracking

As mentioned in Steps 6 and 7 of the basic procedure, backtracking is used but strictly limited. Clearly the optimum can be found, in principle, by full backtracking at all levels, but the running time for such a procedure would be immense. Our backtracking appears to be an effective compromise between power and running time.

Measurements show that, if a gain is to be found at a node, it is usually the first choice made by the procedure (the mean choice number is 1.2 for the first level and 1.8 for the second). Hence, we in fact consider at most only a few contenders for y_1 and y_2 (a maximum of five each at the moment). Experiments indicate that this has no significant effect on the frequency of optimum values, and reduces the running time by a factor of nearly two over considering all y_1 and y_2. Considering only two contenders for each of y_1 and y_2 does cut the effectiveness, although run time is further reduced as well.

Limited experiments with backtracking at the third level indicate a considerable time penalty. Backtracking only at the first level weakens the procedure: although the solutions have quite good costs, comparable to those with more backtrack, the appearance of the optimum is less frequent. (This might be a worthwhile saving for large problems, or for problems where a global optimum is not critical.)

2. REFINEMENTS

THE ALGORITHM AS presented is quite effective, but we have added several refinements that increase its efficacy substantially without significant extra cost.

A. Avoiding Checkout Time

When the procedure produces a tour T of value f at some node t, and is then unable to make further progress at any node, we say that T is a local optimum, and call the time spent after T has been found until all nodes are investigated *checkout time*. If, on a later case, we arrive at the same tour T, there is no point in attempting to find further reduction—this situation has been previously 'checked out.' Thus, to economize on running time, when no improvement can be made at a node, the solution obtained is tested to see if it is the same as a previous one; if so, the checkout time can be avoided. This is typically 30 to 50 per cent of running time.

B. Lookahead

The procedure described selects a link y_i without regard to the size of the x_{i+1}, which will thus have to be broken in the next stage. If this x_{i+1} is actually a 'good' link, i.e., one low in cost, the result is often a long fruitless chase only to discover that breaking x_{i+1} was wrong, and hence that y_i was a poor choice.

To overcome this weakness, a restricted lookahead is added to the procedure: in all steps where a y_i is chosen, the choice is made not on the basis of $|y_i|$ alone, but by trying to maximize $|x_{i+1}|-|y_i|$. When i is 1 or 2, this changes the order of consideration of links, which may have an effect on the solution. (*See* the next paragraph.) In subsequent stages ($i \geqq 3$), it means choosing the y_i corresponding to a maximum value of $|x_{i+1}|-|y_i|$. One might assume that we should maximize $|x_{i+1}|-|y_i|$ over all possible y_i's, but after experimenting we rejected this as too time-consuming. Currently we use the largest $|x_{i+1}|-|y_i|$ found among the five smallest (and available) $|y_i|$, which appears to be just as effective and much faster.

There is another and more subtle reason why the lookahead feature is desirable. If we decide to join links solely on a nearest-neighbor-first basis, we commit ourselves on local information, the length of the link. This means that whenever there is a choice between joining this link instead of some other, the nearest-neighbor link is always chosen, and if this is wrong, there is no way to avoid it. But if we use $|x_{i+1}|-|y_i|$ as the criterion, y_i may or may not be chosen, depending on x_{i+1}, which is a function of the current state of the tour, and thus more global in nature. This gives an added degree of flexibility to the procedure, and improves our chances of getting the right answer.

C. Reduction

Once a number of locally optimum tours have been found by the procedure thus far described, we observe that certain links occur in all of them. Since much

of the time spent by the procedure thereafter is essentially duplication of previous effort by breaking these good links repeatedly, we have devised a reduction feature that attempts to use this information to guide further search (and incidentally save running time).

A small number (currently between two and five) of distinct local optima are found. (If we cannot find two distinct tours in a reasonable number of tries, we take this as *prima facie* evidence that the single solution will not be improved.) Record all links common to all of these solutions. Now ('post-reduction') continue finding locally optimum solutions, with the additional restriction that no link x_i may be broken if it occurs in this set of good links. As new locally optimum tours are found, continue recording the intersection.

The effect of reduction is felt in several ways. The most obvious is a dramatic decrease in running time—a factor of five is typical—because there are so few situations left to evaluate. The reduction procedure has 'trimmed the tree' of possible cases. If the locally optimum tours have many links in common (two distinct local optimum tours usually have 85 per cent in common, and even 7 or 8 will still have 60–80 per cent), then fewer possibilities need be evaluated by the algorithm. (Remember that the good links correspond to the smallest values of y_i's, and so, when these are eliminated by reduction, the gain criterion becomes a more powerful eliminator.)

However, this time saving is not our only purpose in introducing reduction. Rather, reduction is a means of *directing search* among a set of otherwise indistinguishable cases. In effect, we are saying that certain cases may be rejected out of hand because they involve breaking links that have shown up in too many good tours to be accidental; breaking such links is likely to be wrong, and we should try something else.

But reduction obviously biases the solutions we get. Suppose the optimum does not occur while we are finding prereduction tours. Then the basis of reduction contains only suboptimal solutions, and the postreduction procedure may not be able to break out of this to actually get the optimum. Thus we modify the reduction slightly—no x_i may be broken *at level 4 or deeper* if it occurs in the intersection of reduction tours; we may still break links as desired at levels 1, 2, and 3. This 'partial reduction' gives us some flexibility to recover from a bad choice of reduction, but retains much of the path-directing and time-saving properties of full reduction.

Since the procedure uses as the basis of reduction the intersection of all tours seen so far, there is a certain convergence effect. If we get bad tours to start with, but later get the optimum, from then on the optimum should occur more frequently (although run time will also increase slowly as the intersection becomes smaller).

D. Nonsequential Exchanges

As we observed earlier (*see* Fig. 2), some configurations cannot be improved by changing a *connected sequence* of links. The procedure is augmented by a limited defense against this situation. After a locally optimum tour has been found, we test, among the links of the tour that may be broken (as determined by the reduction procedure), whether we can make a further improvement by an exchange such

as shown in Fig. 2. Since the number of links in the tour that differ from links in the reduction is small, this test is rapid. Whether this process yields the desired improvement varies substantially from problem to problem: in some situations it converts the majority of nonoptimum solutions into optimum, while in others it has no effect whatsoever. Running time is under 50 milliseconds per unique tour in our implementation, so it is not very expensive insurance.

3. COMPUTATIONAL RESULTS

ONE MUST DEFINE the power of a heuristic method on the basis of the statistical distribution of answers produced over a spectrum of problems. There are several interrelated measures to consider: (a) How frequently does the procedure, started from a random initial configuration, come up with an optimum tour? (b) What is the number and distribution of distinct locally optimum solutions it produces? (c) How confident are we that it will produce at least one occurrence of the optimum in a sequence of trials? [This is closely related to (a), but is also based on (b).]

We are able to measure (a) directly for our prereduction process; the probability of obtaining optimum solutions in a single trial is close to 1 for small-to-medium problems, say up to the 42-city problem.[3] It drops slowly to about 0.2 to 0.3 for 100-city problems, although individual cases may be substantially higher. The postreduction frequency is, of course, biased to a certain extent by the prereduction tours, but is generally higher than prereduction, and the run time is substantially improved.

For most of the smaller problems in the literature (up to 57 cities) the optimum answers are known; this is not the case for larger problems. This is where (b) is important. If the procedure produces, over a large number of trials from completely random starting configurations, a relatively small set of distinct answers, and if the lowest value occurs with sufficient frequency, we have a strong statistical justification for stating that the lowest value is in fact the optimum solution. For instance, in the 48-city problem[4] on over 200 trials, we obtain *only four* distinct solutions (11461, 11474, 11508, 11574) with relative frequencies 0.3, 0.3, 0.3, 0.1. This set is small, and the minimum value occurs with high frequency, so we are confident in saying that 11461 is optimum (which it is[5]). Similarly, on the 57-city problem,[6] we obtain only eight distinct solutions; the minimum-cost solution occurs 27 per cent of the time, and the three best account for 75 per cent of all solutions.

The overall procedure has always obtained the optimum (or best known) solution if allowed to run for a time proportional to about n^2, although, as effectiveness diminishes with n, this time will increase for larger problems. Thus we can say, for instance, that with under a minute of processing we will *solve* (i.e., find an optimum tour for) a 50-city problem with over 99 per cent confidence (based on 8 seconds per case and 0.5 probability of an optimum on a single trial); with 3–4 minutes we will solve a 100-city problem, and so on.

We have evaluated the procedure on a spectrum of problems: (a) 'classical' problems published by others who have studied the traveling-salesman problem;

(b) problems obtained by placing points randomly in Euclidean 2-space; (c) problems obtained by placing random distances into a distance matrix; and (d) a few real problems derived from controlling automatic drilling equipment.

Problem sizes ranged up to 110 points, where computer-storage limitations have restricted further experiments. Work is in progress to modify the storage-management algorithm so bigger problems may be tackled.

(a) *Classical problems.* It is a measure of the difficulty of the traveling-

TABLE I

SOLUTIONS OF CLASSICAL PROBLEMS

Size	Source	Frequency of occurrence of optimum		Running time per tour (sec., GE635)		Average number of distinct solutions[c]
		Pre-	Post- [a]	Pre-	Post-	
20	[2]	1.00	—	0.8	—	1
25	[4]	1.00	—	1.3	—	1
33	[6]	1.00	—	1.6	—	1
42	[3]	1.00	—	2.6	—	1
46	[5, Tutte]	0.95	—	0.9	—	—
48	[4]	0.30	0.50	2.4	2.3	3
57	[6]	0.27	0.43	8.2	3.8	3
64	8×8 Knight's Tour, var. 1	1.00	0.95[b]	2.0	3.1	—
64	8×8 Knight's Tour, var. 2	1.00	1.00[b]	2.1	2.5	—
64	8×8 Knight's Tour, var. 3	0.51	0.50	2.8	2.8	—
100	[9, ✻24]	0.56	0.63	17	5.5	3.5
100	[9, ✻25]	0.25	0.35	23	9.5	4.5
100	[9, ✻26]	0.52	0.65	17	5.7	3
100	[9, ✻27]	0.50	0.60	25	6.9	3.5
100	[9, ✻28]	0.23	0.30	30	11	5

[a] Frequency of optimum based on three reduction tours. Optimum values conjectured for 100-city problems: ✻24 and ✻27 agree with reference 9; ✻25, ✻26, and ✻28 are better.

[b] Optimum knight's tours are not unique.

[c] Out of 20 trials.

—not applicable.

salesman problem that the classical literature is largely based on small (less than 60 cities) Euclidean-distance problems, and that results are often obtained only at considerable computing expense. Many authors are, of course, more interested in obtaining *the* optimum answer to a problem, rather than a spectrum of very good answers, among which is likely to be found the optimum. (*See* reference 5, for instance.)

Because of such theoretical work, the optimum answers to a small family of problems are known and these provide an interesting test of any procedure. We have studied the problems listed in Table I. These are all easy, in the sense that we have never failed to get the optimum answer in a small number of runs. We obtained better solutions to three of the five 100-city problems (✻25: 22148 vs 22193;

#26: 20749 vs 20852; #28: 22068 vs 22115) mentioned in reference 9, without the need for the 'man' part of man-machine interaction, and with substantially less machine time.

An interesting special problem is the Tutte graph.[5] This 46-city graph has no Hamiltonian circuit, but does have a Hamiltonian path, so, if the distances are set to 0 where there is an edge, and $k>0$ otherwise, an optimum traveling-salesman solution will have cost k. This problem is especially easy for our procedure: the frequency of optimum solutions is close to 100 per cent, and run time is substantially lower than for other problems of the same size.

We have also included three variants of the 8×8 knight's-tour problem in this list. Variant 1 has distance matrix entries 1, 2, $\cdots$, 6 depending on the number of knight's moves necessary to get from one square to another. Variant 2 sets distances between squares a knight's move apart to 1, and all other distances to relatively large random numbers. Variant 3 sets cells a knight's move apart to distance 100, all others to 101.

TABLE II

SOLUTIONS FOR RANDOM POINTS ON THE UNIT SQUARE

Size	Avg. Freq. of optimum[a]	Avg. time/case (sec)		Avg. distinct solutions post reduction
		Pre[b]	Post[c]	
30	0.85	2.4	0.65	1.2
50	0.89	8.2	1.9	2.0
70	0.70	10.5	3.5	2.2
90	0.19	22	11.5	8.5
110	0.21	24.5	12.2	8.5

[a] Occurrences of optimum averaged over at least four distinct problems.
[b] Includes checkout time.
[c] Based on three distinct reduction tours.

Variants 1 and 2 are similar in result—very easy, with essentially 100 per cent success. Variant 3 is harder—run time increases and the frequency of optimum solutions drops to barely 50 per cent.

(b) *Random points on the plane.* The second class of traveling-salesman problems is generated by randomly scattering points on the unit square (x and y coordinates generated uniformly and independently).

These also appear to be generally easy, as Table II indicates, although individual cases vary considerably. For any specific problem, we have done a sufficient number of trials (20–40) to be confident that we have, if not the optimum, at least a good near-optimum.

The frequency of obtaining the optimum solutions is so high that it appears that we can practically guarantee optimality with a modest amount of computer time, for problems of all sizes tested.

Further, the running times scale well enough that (if we organize storage properly) 200- to 300-city problems should also be well within reach.

(c) *Random entries in the distance matrix.* These problems are generated by placing random numbers uniformly distributed on $[1,100n]$ into a symmetric n by n

TABLE III
SOLUTIONS FOR RANDOM ENTRIES IN THE DISTANCE MATRIX

Size	Avg. freq. of optimum[a]	Avg. time/case (sec)		Avg. distinct solutions post reduction
		pre[b]	post[c]	
30	0.46	1.9	1.0	3.6
50	0.68	6.2	2.6	4.6
70	0.17	12.5	6.6	11
90	0.13	19.5	11.1	12.5
110	0.09	30	17	18

[a] Occurrence of optimum averaged over at least four distinct problems.
[b] Includes checkout time.
[c] Based on three distinct reduction tours.

distance matrix, yielding a nonmetric space problem. These are, in general, more difficult than metric problems, and individual variation is higher, as may be seen by comparing Table III to Table II.

(d) *A real problem.* It has often been observed that the problem of routing a

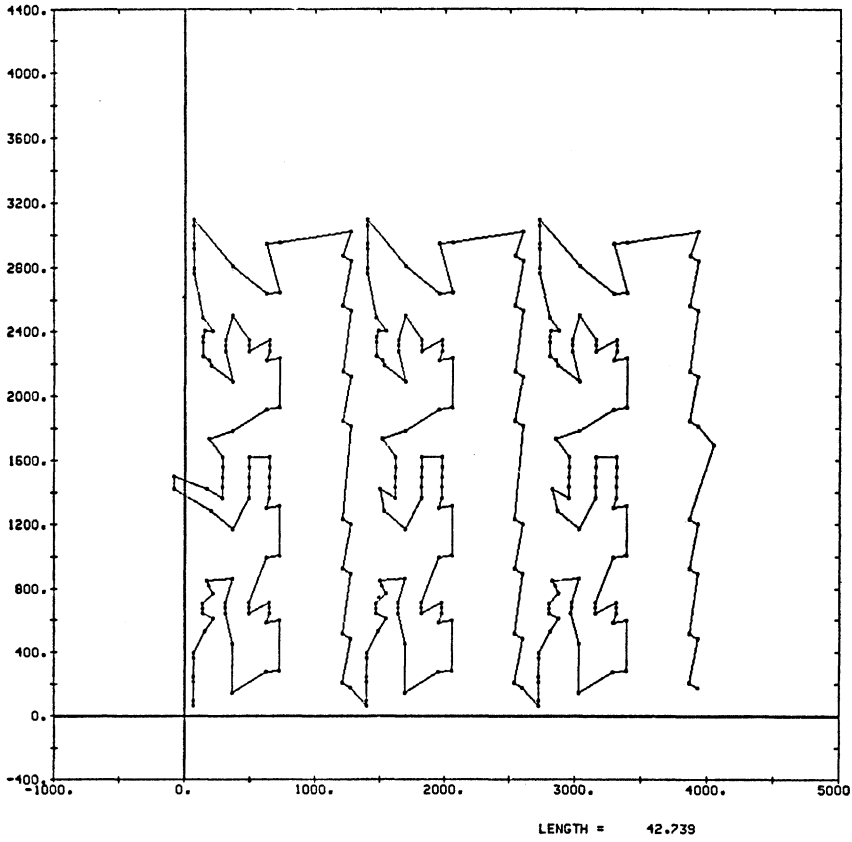

Fig. 6. The hand solution for the 318-point problem.

numerically controlled drilling machine efficiently through a set of hole positions is a traveling-salesman problem, but if drilling time outweighs travel time, there is no particular advantage to any optimization. However, as we were experimenting, we were given a problem in which drilling is not done mechanically, but by a pulsed laser, and hence almost the entire cost *is* in travel time. (We are indebted to R. HABERMANN for providing us with the problem and his solution for comparison.)

The actual problem consists of three identical sets of 105 points, plus three

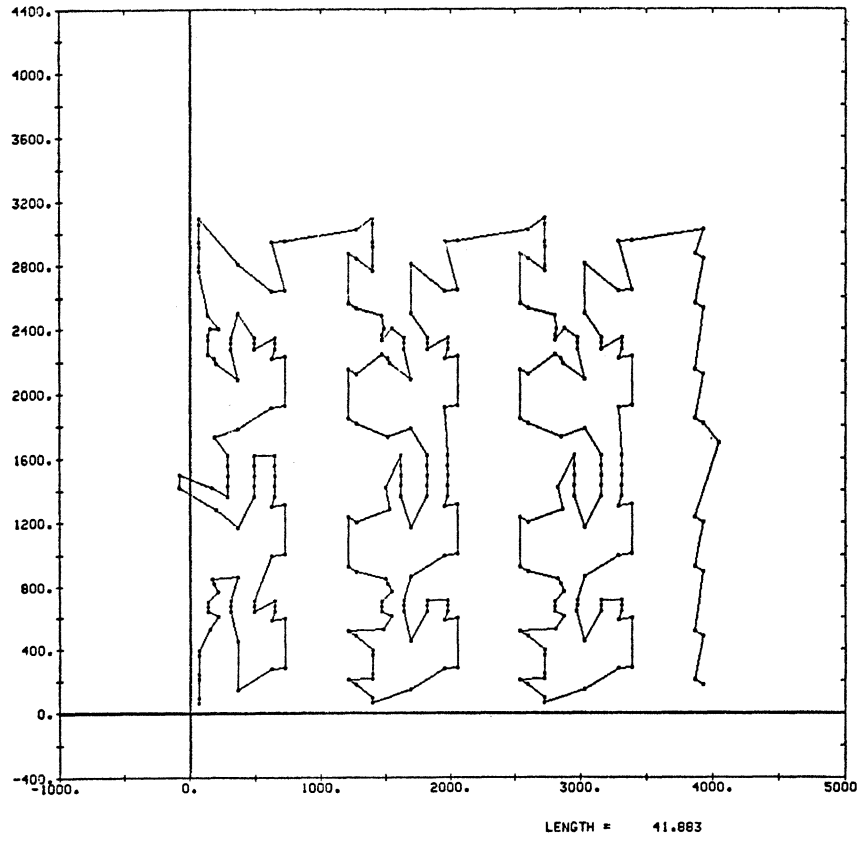

Fig. 7. The machine solution for the 318-point problem.

stragglers, as shown in Figs. 6 and 7. Although there is a considerable regularity, the problem appears to be comparable to other Euclidean 2-space problems in both run time and solution quality. The tour of Fig. 6 was generated by hand (R. Habermann) by splicing together the three obvious pieces, and forcing the two end nodes together with an edge (not shown) of length zero. Figure 7 is a tour we obtained by joining several pieces of about 100 points, each optimized by our procedure. This tour is quite different in character from the first, and about 0.85 inch shorter (out of 42 inches) assuming the endpoints are as shown. (We make no claim that the result for 318 points is optimum—there are many ways to join pieces. Later work will investigate the effect of treating this problem as a unit.)

TABLE IV

COORDINATES FOR THE 318-POINT PROBLEM

1	71	63	36	630	992	71	142	2244
2	71	94	37	732	1000	72	315	2276
3	370	142	38	1276	1197	73	496	2276
4	1276	173	39	1213	1228	74	654	2276
5	1213	205	40	205	1276	75	654	2315
6	69	213	41	630	1299	76	496	2315
7	69	244	42	732	1307	77	315	2315
8	630	276	43	654	1362	78	142	2331
9	732	283	44	496	1362	79	315	2346
10	69	362	45	291	1362	80	496	2346
11	69	394	46	654	1425	81	654	2346
12	370	449	47	496	1425	82	142	2362
13	1276	480	48	291	1425	83	157	2402
14	1213	512	49	173	1417	84	220	2402
15	157	528	50	291	1488	85	142	2480
16	630	583	51	496	1488	86	370	2496
17	732	591	52	654	1488	87	1276	2528
18	654	638	53	654	1551	88	1213	2559
19	496	638	54	496	1551	89	630	2630
20	314	638	55	291	1551	90	732	2638
21	142	638	56	291	1614	91	69	2756
22	142	669	57	496	1614	92	69	2787
23	315	677	58	654	1614	93	370	2803
24	496	677	59	189	1732	94	1276	2835
25	654	677	60	1276	1811	95	1213	2866
26	654	709	61	1213	1843	96	69	2906
27	496	709	62	630	1913	97	69	2937
28	315	709	63	732	1921	98	630	2937
29	142	701	64	370	2087	99	732	2945
30	220	764	65	1276	2118	100	1276	3016
31	189	811	66	1213	2150	101	69	3055
32	173	843	67	205	2189	102	69	3087
33	370	858	68	189	2220	103	220	606
34	1276	890	69	630	2220	104	370	1165
35	1213	921	70	732	2228	105	370	1780

Second 105-point subset—add 1331 to x coordinate; add 2662 for third set. Three 'stragglers' have coordinates (−79, 1417), (−79, 1496), (4055, 1693). Coordinates are in milli-inches.

Table IV gives the coordinates for those who would like to try their luck on a big problem.

Run-Time Analysis

Run times for both prereduction and postreduction trials are included in Tables I–III. Plotting these times for the various types of problems shows that, for both

the Euclidean problems and the nonmetric problems, run time grows approximately as $n^{2.2}$; postreduction times are smaller by a factor of two to three. In absolute terms, 100-city problems average slightly over 20 seconds per case (Fortran A, GE635) before reduction, and 8 seconds after.

The functional growth can be explained as follows. At each node we search for a sequence of improvements; on the average, the depth of this search is a slowly growing function of n, say $d(n)$. (Depth is quite close to n in the early stages, but rapidly becomes small.) However, as each potential exchange in the sequence is found, we must alter the internal representation of the tour to keep it as a tour; this costs a factor of n, so a single probe at a single node costs $nd(n)$. Since the backtrack at levels 1 and 2 requires a bounded number of probes per starting node, this contributes only a constant factor, which depends on the amount of backtracking used.

These probes are repeated for all n nodes, until no gain can be found. Generally we enter the checkout phase before even n nodes have been considered, so there is at worst a factor of n accumulated here. Thus, overall growth is $n^2d(n)$, somewhat worse than n^2. Postreduction, the number of alternatives decreases for each part of the procedure, so the constant of proportionality is smaller. In addition, $d(n)$ should also decrease, but we have not tried to isolate this effect.

4. EXTENSIONS AND CONCLUSIONS

SEVERAL PROBLEMS RELATED to the basic traveling-salesman problem may be solved by simple manipulations of the distance matrix. For example, we may force a group of cities to be together in a tour by making the distance between them very small; similarly we can force them apart by artificially large distances. Thus we can solve shortest Hamiltonian path problems by joining the two end points with a small distance. (This was done for the 318-point problem discussed in the previous section.)

Large Problems

A study of optimum and near-optimum tours for several problems shows that, as might be expected, most cities are connected to quite nearby neighbors. The typical distribution is one-half nearest neighbors, and a sharp drop-off with only one or two links as far out as the 15th-nearest neighbor. This suggests we can economize on storage without detriment to solution quality by storing only some fraction of the nearest neighbors of each city, rather than the entire distance matrix. Since our basic procedure contains several tradeoffs between effectiveness and running time, by sacrificing some effectiveness, we can decrease running time for larger problems. For instance, we can cut down on the number of alternatives allowed at levels 1 and 2; if these are set to 1 or 2, we have in fact a modified algorithm that is still quite effective.

REFERENCES

1. M. Bellmore and G. L. Nemhauser, "The Traveling-Salesman Problem: A Survey," *Opns. Res.* **16,** 538–558 (1968).
2. A. Croes, "A Method for Solving Traveling-Salesman Problems," *Opns. Res.* **5,** 791–812 (1958).
3. G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, "Solution of a Large-Scale Traveling-Salesman Problem," *Opns. Res.* **2,** 393–410 (1954).
4. M. Held and R. M. Karp, "A Dynamic-Programming Approach to Sequencing Problems," *J. SIAM* **10,** 196–210 (1962).
5. ——— and ———, "The Traveling-Salesman Problem and Minimum Spanning Trees, Part I," *Opns. Res.* **18,** 1138–1162 (1970); "Part II," *Math. Prog.* **1,** 6–26 (1971).
6. R. L. Karg and G. L. Thompson, "A Heuristic Approach to Solving Traveling-Salesman Problems," *Management Sci.* **10,** 225–247 (1964).
7. R. M. Karp, "Reducibility among Combinatorial Problems," Computer Science Tech. Rpt. #3, University of California, Berkeley, April 1972.
8. B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *BSTJ* **49,** 291–308 (1970).
9. P. Krolak, W. Felts, and G. Marble, "A Man-Machine Approach toward Solving the Traveling-Salesman Problem," *CACM* **14,** 327–334 (1971).
10. M. J. Krone, "Heuristic Programming Applied to Scheduling Problems," Ph.D. Thesis, Princeton University, September 1970.
11. S. Lin, "Computer Solutions of the Traveling-Salesman Problem," *BSTJ* **44,** 2245–2269 (1965).
12. K. Steiglitz, P. Weiner, and D. J. Kleitman, "The Design of Minimum-Cost Survivable Networks," *IEEE Trans. on Circ. Theory,* **CT-16,** 455–460 (1969).