Computer Solutions of the Traveling Salesman Problem

By SHEN LIN

(Manuscript received August 18, 1965)

Two algorithms for solving the (symmetric distance) traveling salesman problem have been programmed for a high-speed digital computer. The first produces guaranteed optimal solution for problems involving no more than 13 cities; the time required (IBM 7094 II) varies from 60 milliseconds for a 9-city problem to 1.75 seconds for a 13-city problem. The second algorithm produces precisely characterized, locally optimal solutions for large problems (up to 145 cities) in an extremely short time and is based on a general heuristic approach believed to be of general applicability to various optimization problems. The average time required to obtain a locally optimal solution is under $30n^3$ microseconds where n is the number of cities involved. Repeated runs on a problem from random initial tours result in a high probability of finding the optimal solution among the locally optimal solutions obtained. For large problems where many locally optimal solutions have to be obtained in order to be reasonably assured of having the optimal solution, an efficient reduction scheme is incorporated in the program to reduce the total computation time by a substantial amount.

I. INTRODUCTION

The traveling salesman problem may be stated as follows: "A salesman is required to visit each of the n given cities once and only once, starting from any city and returning to the original place of departure. What route, or tour, should he choose in order to minimize the total distance traveled?" Instead of distance, other notions such as time, cost, etc., can be considered as well. In this paper, we shall use the term "cost" to represent any such notion.

Mathematically, the problem may be stated in the following two equivalent ways:

(1) Given a "cost matrix" $D = (d_{ij})$, where d_{ij} = cost of going from

city i to city j, $(i, j = 1, 2, \cdots, n)$, find a permutation $P = (i_1, i_2, i_3, \cdots, i_n)$ of the integers from 1 through n that minimizes the quantity

$$d_{i_1 i_2} + d_{i_2 i_3} + \cdots + d_{i_n i_1}.$$

(2) Given a "cost matrix" D as above, determine x_{ij} which minimizes the quantity $Q = \sum_{i,j} d_{ij} x_{ij}$ subject to

(a) $x_{ii} = 0$

(b) $x_{ij} = 0, 1$

(c) $\sum_i x_{ij} = \sum_j x_{ij} = 1$

and

(d) for any subset $S = \{i_1, i_2, \cdots, i_r\}$ of the integers from 1 through n,

$$x_{i_1 i_2} + x_{i_2 i_3} + \cdots + x_{i_{r-1} i_r} + x_{i_r i_1} \begin{cases} < r & \text{for } r < n \\ \leqq n & \text{for } r = n. \end{cases}$$

The second version is a formulation of the traveling salesman problem as a linear program and hence the problem may be solved as such. However, the number of constraints becomes astronomical even for relatively small n. Dantzig, Fulkerson, and Johnson[1] have given a linear-programming approach to the symmetric $(d_{ij} = d_{ji})$ traveling salesman problem that considers only part of the required linear constraints and have found the technique effective in several cases.

Since we have only a finite number of possible tours to consider $(\frac{1}{2}(n-1)!)$, the problem is really to obtain a reasonably efficient algorithm for finding an optimal solution. Certain algorithms employing branch and bound techniques have been tried and appear to be efficient for some problems; however, the computation time involved is unpredictable and increases very rapidly with n. Numerous authors have tried different techniques to obtain "near-optimal" solutions by a series of approximations and for specific problems were able to prove optimality of their solutions. For any conjectured optimal solution, however, the proof for optimality is dependent upon inspectional work which is usually heuristic in nature, and is certainly highly problem dependent, thus making it difficult to program for a computer.

Two algorithms for solving the (symmetric distance) traveling salesman problem have been programmed for a high-speed digital computer. The first algorithm, called k-length string optimization, is discussed in detail in Appendix A. It produces guaranteed optimal solutions for problems involving no more than 13 cities; the time required* varies

* IBM 7094 II.

from 60 milliseconds for a 9-city problem to 1.75 seconds for a 13-city problem. The algorithm is a slight modification of that given by Held and Karp.[2] However, we achieve a significant reduction in computation time by taking advantage of the fact that the distance matrix is symmetric. Due to the limitation on the size of the problem it can effectively handle, we find that it is not as useful as the second algorithm which we shall discuss below. The second algorithm (implemented by a copyrighted program) produces precisely characterized locally optimal solutions for large problems (up to 145 cities) in an extremely short time and is based on a general heuristic approach believed to be of general applicability to various optimization problems. The average time required per locally optimal solution is under $30n^3$ microseconds where n is the number of cities involved. Repeated runs on a problem from random initial tours result in a high probability of finding the optimal solution among the local optimum solutions obtained. For large problems where many locally optimal solutions have to be obtained in order to be reasonably assured of having the optimal solution, an efficient reduction scheme is incorporated in the program to reduce the total computation time by a substantial amount.

II. λ-OPTIMALITY

Before we describe the second algorithm, we first introduce the concept of λ-optimality. This serves to classify tours into a descending chain of classes possessing increasingly stronger necessary conditions for optimality. As we shall see later, this forms the basis for the construction of our second algorithm.

From the point of view of graph theory, we may consider the n cities as vertices of a nondirected complete graph, and the entries d_{ij} of the distance matrix real numbers assigned to links u_{ij} connecting city i to city j. A permutation $P = (i_1, i_2, \cdots, i_n)$ representing a tour may be considered as a collection of n links $u_{i_1 i_2}, u_{i_2 i_3}, \cdots, u_{i_n i_1}$ forming a Hamiltonian circuit, and the quantity $C = d_{i_1 i_2} + d_{i_2 i_3} + \cdots + d_{i_n i_1}$ the cost associated with the tour.

For convenience, let us say a link u_{ij} is *admissible* if there is an optimal tour containing it. All other links are *inadmissible*. A set of links is said to be an *admissible set* if there exists an optimal tour containing all the links in the set. The *index* of a tour is the maximum number of links which the tour has in common with an optimal tour, i.e., the maximum number of links in the tour which form an admissible set.

We define λ-optimality of tours as follows:

Definition: A tour is said to be λ-*optimal* (or simply λ-opt) if it is im-

possible to obtain a tour with smaller cost by replacing any λ of its links by any other set of λ links.

We list below a few theorems concerning λ-optimality. Proofs are omitted since they are all fairly obvious. In Appendix D some interesting unsolved problems concerning λ-optimality will be discussed.

Theorem 1: Let T be a tour which is λ optimal with index k. Then either T is optimal or $k < n - \lambda$.

Theorem 2: Any tour is 1-optimal.

Theorem 3: The following properties of a tour are equivalent:

(a) The tour is 2-optimal.

(b) The tour is optimal relative to inversion; where by inversion we mean reversing the order of a set of neighboring cities in the tour.

(c) The tour does not intersect itself (in a generalized sense for non-Euclidean distance matrices).

Theorem 4: A tour is optimal if and only if it is n-optimal.

Theorem 5: Let C_λ denote the set of all λ-optimal tours, then $C_1 \supset C_2 \supset \cdots \supset C_n$. In other words, a λ-optimal tour is also λ'-optimal for $\lambda' < \lambda$.

Note that the well-known theorem, which states that an optimal tour does not intersect itself, is contained in Theorems 3, 4, and 5. ($C_n \subset C_2$).

III. THE SECOND ALGORITHM

In his paper, "A Method for Solving Traveling Salesman Problems", G. A. Croes[3] applied a simple transformation, called "inversion" to transform a trial solution into another with smaller costs, iterating until no further inversions are desirable. Then he gave a method for deriving the optimal solution from the inversion free solution obtained. He pointed out, however, that the final adjustment procedures are difficult to program for a computer as they involve mostly inspectional work. For large problems, it seems doubtful whether a human being can exhaustively carry the computations through or even whether a computer program based upon those techniques will be feasible or efficient.

Putting aside the final adjustment procedures, we ask if there are other simple transformations which are stronger than the inversions. Since the inversion-free tours are just the 2-opt tours, we decided to write a computer program to produce 3-opt tours. As it turns out, the 3-opt tours are very much stronger than the inversion-free tours in the sense that (1) every 3-opt tour is inversion free, (2) the average tour

cost is considerably less, and (3) the probability of an optimal solution showing up as a 3-opt tour is significantly higher than that of 2-opt tours. Experimenting with a program producing 4-opt tours, we also find that we spend a great deal more computation time in producing 4-opt tours while not increasing noticeably the probability that it is optimum. Computational results on many problems support the claim that we have found a really efficient way of attacking the traveling salesman problem by generating as many 3-opt tours from random initial tours as we can afford time-wise and then choosing the best among the 3-opt solutions as our "conjectured solution". The merits of this heuristic approach based on probability as compared to the usual approach of using further complicated refinements to transform locally optimal solutions into global optima will be discussed later in the paper.

IV. THE GENERAL APPROACH

Since we have found that a 3-opt tour has a nontrivial probability of being optimal, we make, for a given problem, an estimate of this probability* (of success) P_s and produce from our program k 3-opt tours (not necessarily distinct) from random initial starts. We choose k so that $1 - (1 - P_s)^k$ is as close to 1 as we desire. Since the running time in obtaining each 3-opt tour is reasonable (25 to $30n^3$ microseconds where n is the number of cities), we can indeed afford the luxury of making k large. For example, a 30-city problem can reasonably be expected to be "solved" in 75 seconds with $k = 100$. At any rate, the best of the k locally optimal solutions, even though it may not actually be the best, will be close enough to the best solution as to offer a satisfactory answer for most practical problems arising in actual applications. Also, a large set of "satisfactory" locally optimal solutions may give an engineer more flexible choice of a solution that he may use satisfying further nonessential but nevertheless desirable features which may be hard to program.

When the number of cities involved is rather large, say >30, the number of locally optimal solutions that needs to be generated in order to be reasonably assured of having the optimal solution will be very large, as is expected. Incorporated in the program, is a reduction scheme whereby information gained from an initial set of locally optimal solutions is used to reduce the size of the problem, thereby decreasing sub-

* This probability, in general, depends on the size and nature of the problem. From the statistics collected after running many problems, we shall give a heuristic estimate in Appendix C.

stantially the time involved in generating additional locally optimal solutions.

A brief description of the computer program to produce 3-opt tours is given in Appendix B. We mention here an alternate characterization of a 3-opt tour which is more graphic and which we use in our program. A tour T is said to be *optimal relative to inversion and insertion* if, for every k, no section of k consecutive cities in T, say $(i_{\alpha+1}, i_{\alpha+2}, \cdots, i_{\alpha+k})$, can be removed from T and reinserted (as is, or inverted) between any two consecutive remaining cities to produce a tour of lesser cost. We prove the following:

Theorem 6: A tour T is optimal relative to inversion and insertion if and only if it is 3-optimal.

Proof: A tour T is not 3-optimal, if and only if there exists 3 links, say u_{ij}, u_{kl}, u_{mn} which may be exchanged by 3 other links say u_{im}, u_{jl} and u_{nk}, (as in Fig. 1, other possibilities are similar) to form a tour of lesser cost. From Fig. 1, we see that the section from m to l may be inserted between i and j and hence the tour is not optimal relative to inversion and insertion.

V. GENERAL DESCRIPTION OF THE METHODS USED TO PRODUCE 3-OPT TOURS

In the process of obtaining 3-opt tours from a random initial tour, the basic operation consists of determining whether any section of length k in the present tour can be inserted (as is, or inverted) between two other neighboring cities so as to decrease the tour cost. This was proved in Section IV (Theorem 6) to be equivalent to exchanging three links in

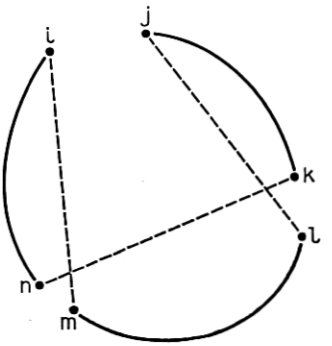

Fig. 1 — Proof of Theorem 6.

the tour for three other links. Once an improvement is found and made, we treat the resulting tour as our initial tour and iterate the process. The process terminates with a locally optimal (3-opt) tour when improvement cannot be further achieved. We call the portion of the computation from the time we made the last improvement to the verification that no further improvement can be achieved by this algorithm the "check-out period." The time involved in the "check-out period" is proportional to $\binom{n}{3}$. For different random initial tours however, the number of improvements may vary and the time it takes to find each improvement also varies. This accounts for some variation in the computation time for the individual locally optimal solutions. However, it turns out that the average computation time for a set of 10 or more cases is uniformly around $50n^3$ microseconds, which is further reduced by the techniques discussed below.

From our experiments, we find that links used in a locally optimal solution are often exchanged in and out many times in the improvement process, and this tends to increase the computation time considerably. Two methods are incorporated in the program to reduce this. First, after each improvement, the improved tour $(t_1', t_2', \cdots, t_n')$ is further perturbed by a rotation (see Appendix B) so as to prevent the new links just inserted from being removed again too soon. Secondly, the following special feature making use of locally optimal tours previously obtained is used: After m 3-opt tours $T_1, T_2, \cdots, T_m$ $(m = 1, 2, \cdots)$ are generated, consider the set S consisting of the union of the links found in $T_1, T_2, \cdots, T_m$. In the process of obtaining the $m + 1$th 3-opt tour, we systematically break off 3 links $u_{t_1t_n}, u_{t_kt_{k+1}}, u_{t_jt_{j+1}}$ in $(t_1, t_2, \cdots, t_n)$ to see if they can be replaced by 3 other links so as to form a tour of lesser cost. In the algorithm (see Appendix B for details), $u_{t_1t_n}$ is held fixed while k goes from 1 to $n - 3$, coupled with each j going from $k + 1$ to $n - 1$. We skip this sequence of tests for possible improvements altogether if $u_{t_1t_n} \in S$, and proceed as if all such tests for possible improvements fail. The tour $(t_1, t_2, \cdots, t_n)$ is "rotated" by the substitution $(t_n, t_1, t_2, \cdots, t_{n-1}) \rightarrow (t_1, t_2, \cdots, t_n)$ and the improvement process continues. When no further improvements can be made relative to this special feature, we obtain a tour which we call an "almost 3-opt tour." This almost 3-opt tour is then put through the algorithm without the special feature to obtain a final 3-opt tour.

This process may seem roundabout, but in effect it results in postponing the replacements of links which have occurred in other locally optimal tours until other replacements have been tried. Actually, an

almost 3-opt tour often has most, if not all, of its links in S so that its check-out period is almost negligible. The time required to obtain the final 3-opt tour is also usually quite small. The over-all effect is that we are able to find improvements much sooner and also the number of improvements made in reaching the locally optimal solution is significantly decreased. As a result, for a set of about 10 locally optimal solutions, we are able to reduce the total computation time by at least 40 percent.

VI. THE REDUCTION PROCEDURE

After a certain number, say r, of locally optimal solutions are obtained, consider the set I of links common to all those locally optimal solutions. Intuitively, we feel that since the r 3-opt tours are produced from randomly generated initial tours, any link in I should have a very high probability of belonging to an optimal solution when r is reasonably large. Further, for each problem certain simple features (like some obvious links connecting two cities) of the optimal solution should be reflected in a majority of 3-opt tours so that we expect the set I to be frequently nonempty. Using I, we can reduce the size of the problem as follows: A link u_{ij} is called *basic* if u_{ij} is in I, and a city i is removed if there are two basic links u_{ij}, u_{ij} incident at i. The procedure can of course remove many strings of cities at the same time. If u_{ji_1}, $u_{i_1i_2}$, $\cdots$, $u_{i_pj'}$ are all basic and no other basic links are connected to cities j and j', the string of cities $i_1, i_2, \cdots, i_p$ is removed. We call cities j and j' *corresponding terminals*. If a total of t cities are removed, we then solve for 3-opt tours in the remaining $n - t$ cities. By reassigning artificial link costs to all links $u_{jj'}$, where j and j' are corresponding terminals, we make sure that j and j' will be neighboring cities in the solution to the reduced problem, and hence the string of cities between j and j' which were removed can be reinserted accordingly. This process can be iterated as many times as we please. Note that we tacitly assume an optimal tour contains the cities $j, i_1, i_2, \cdots, i_p, j'$ as a substring. If this is the case, we say that the reduction is *proper*. Otherwise, the reduction is *improper* and the optimal tour will be missed in all future 3-opt tours generated in the same run. However, even if this should happen (rare if r is large or $n \leqq 30$), the best tour obtained usually has a tour cost differing from the optimal tour by an extremely small amount. For large n, several independent runs should be made to guard against the possibility of an improper reduction.

When the number of cities involved is fairly large, this reduction

procedure is very effective in reducing computation time required to obtain the optimal solution as the results given in the next section show. The reduction procedure also provides a large variety of hard problems (involving a smaller number of cities) from which we can learn a great deal statistically about the characteristics of 3-opt tours. We consider those problems harder than randomly chosen problems because they retain essentially the heart of the original larger problem. The probability that a 3-opt tour generated from a reduced problem is optimal relative to the reduced problem is usually lower than the mean probability for random problems of the same size. However, in spite of the fact that improper reduction may occur, the probability that a 3-opt tour generated from a reduced problem is optimal relative to the original problem cannot be decreased. Since problem-size is reduced, more 3-opt tours can be obtained in a given time, and thus the probability of finding the optimal solution in a given amount of computation time is greatly increased.

VII. COMPUTATIONAL RESULTS

7.1 *Twenty and Fewer City Problems*

Six problems whose cities are points (x_i, y_i) generated randomly in a 100×100 square and of sizes ranging from 12 to 20 cities were tested. For these cases, 5 to 10 3-opt tours were generated per problem. It turned out that in each problem all 3-opt tours generated were identical, and the costs of the solutions obtained in all six problems were as good as, or better than, solutions obtained by other methods (3 of which are known to be optimum). It appears that randomly generated problems are easy to solve by our method.

Forty 3-opt tours were generated for the 20-city problem of G. A. Croes,[3] which has a known optimal solution with cost 246. Reduction was used with $r = 8$. Successive stages of reduction reduced the number of cities from 20 to 11, 11, 11, 11, (i.e., no further reduction produced after the second round). The optimal tour appeared 13 times out of 40 and the total computation time used for the 40 3-opt tours was 3.43 seconds. This 20-city problem seems "harder to solve" than most 20-city problems we have encountered.

Many more problems with sizes around 20 cities obtained from the reduction process of larger problems were investigated. Judging from all the results, we believe that we can "solve" any 20 or fewer city problem by our method in (very conservatively) 5 to 10 seconds.

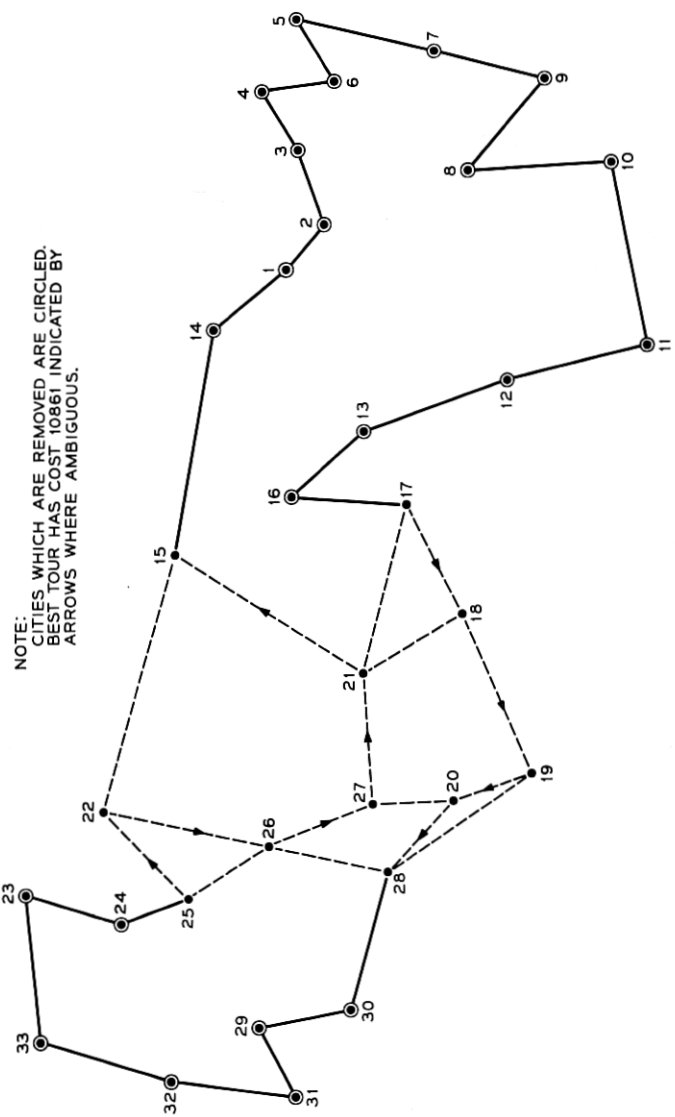

Fig. 2 — 33-City problem showing the link sets I and S after the first, second, and third stages of reduction.

7.2 *The 25-City Problem of Held and Karp*[2]

Forty 3-opt tours were generated with reduction in sets of 10. The reduction procedure reduced the size of the problem from 25 to 10 to 7 and 7. The optimal tour (cost 1711) appeared 26 times out of 40. Total computation time was 5.24 seconds. It is interesting to note that there are 7-city problems produced as a result of our reduction for which 3-opt tours are not necessarily optimal.

7.3 *The 33-City Problem of Karg and Thompson*[4]

This 33-city problem seems very easy to solve by our methods. Fifty 3-opt tours were generated with reduction in sets of 10. The reduction procedure reduced the size of the problem from 33 to 11, 11, 11, and 9. The optimal tour (cost 10,861) appeared 19 times out of 50 and the total computation time was 10.7 seconds. Figs. 2 and 3 illustrate some stages in the reduction process. The solid lines indicate links in the set I, and together with the broken lines, form the set S of all links found in the 3-opt tours generated.

7.4 *The 42-City Problem of Dantzig, Fulkerson and Johnson*[1]

A 42-city problem was solved by Dantzig, Fulkerson and Johnson[1]. The optimal tour has cost 699. Forty 3-opt tours were generated with the optimal tour appearing 11 times. Total computation time was 36.3 seconds. The successive stages of reduction and other pertinent information obtained are given in Table I. Note that d, the number of distinct 3-opt tours obtained per round, decreases, indicating that for smaller problems, there are not too many distinct 3-opt tours and hence the probability that a 3-opt tour is indeed optimal is quite large.

7.5 *The 48-City Problem of Held and Karp*[2]

In Ref. 2, Held and Karp obtained the "best" solution to this 48-city problem with cost = 11,470. D. W. Sweeney (private communication) later found a tour with cost = 11,461. We strongly conjecture that this is indeed the optimal tour and shall consider it as such for the purpose of our work.

Statistics collected on numerous runs indicate that a 3-opt tour has a probability $p \approx 0.05$ of being optimal, with each 3-opt tour obtained in the average computation time of 2.80 seconds.

Without the reduction procedure, we need to spend about 280 seconds to produce 100 3-opt tours if we want a probability of 0.99 of obtaining

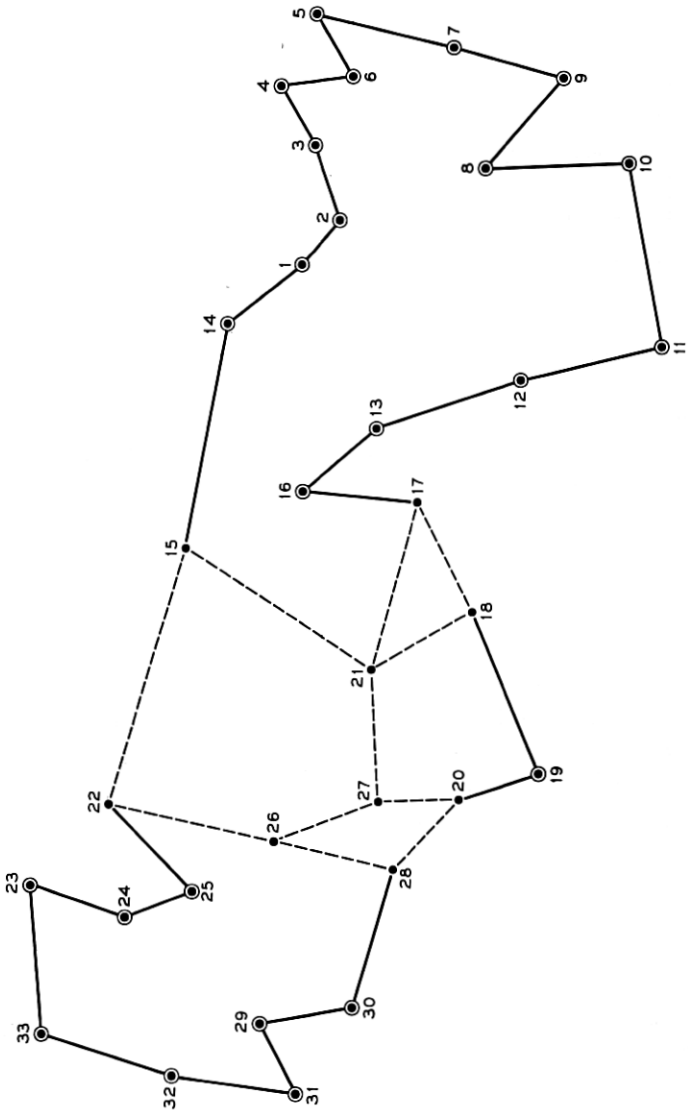

Fig. 3 — As a result of 4th round reduction cities 19 and 25 are also removed.

TABLE I — 42-CITY PROBLEM SUMMARY

SRR	n	r	d	C_M	C_m	$\bar{C}$	t	T
1	42	10	6	734	699	713	1	20.8
2	30	10	5	713	699	704.2	4	28.3
3	24	10	5	713	699	705.9	2	32.2
4	24	10	4	713	699	703.7	4	36.3

SRR: Successive rounds of reduction.
n: Number of cities in reduced problem.
r: Number of 3-opt tours generated per round.
d: Number of distinct 3-opt tours obtained per round.
C_M: Maximum tour cost (for the current round).
C_m: Minimum tour cost (for the current round).
$\bar{C}$: Average tour cost (for the current round).
t: Number of occurrences of the best tour in the current round.
T: Total time of computation in seconds (accumulated).

the optimal solution. With reduction, setting $r = 10$, we obtained the optimal solution 21 times in a total computation time of 63 seconds, as shown in Table II. When d drops below $r/2$, we consider the resulting reduction too binding in evaluating the heuristic probability p. For example, from this particular run, we count 4 out of 50 instead of 21 out of 100 as the frequency of occurrences of the optimal tour.

TABLE II — A TYPICAL 48-CITY RUN

SRR	n	r	d	C_M	C_m	$\bar{C}$	t	T
1	48	10	10	11,887	11,470	11,666.8	1	28.4
2	37	10	8	11,989	11,474	11,622.9	1	41.4
3	34	10	8	11,716	11,461	11,592.9	1	51.0
4	27	10	5	11,704	11,461	11,566.2	1	56.1
5	20	10	5	11,556	11,461	11,511.7	2	58.3
6	18	10	5	11,556	11,470	11,527.2	1	60.0
7	17	10	2	11,556	11,461	11,508.5	5	61.4
8	10	10	2	11,556	11,461	11,480	8	62.0
9	10	10	2	11,556	11,461	11,537	2	62.5
10	10	10	2	11,556	11,461	11,537	2	63.0

7.6 *The 57-City Problem of Karg and Thompson*[4]

In Ref. 4, Karg and Thompson introduced a 57-city problem and found by their method tours with costs of 12,986 and 12,985. In Ref. 5, Reiter and Sherman developed a family of algorithms and found two tours which are better with costs 12,955 and 12,967.* Our program also

* The next best tour we obtained is one with cost 12,966. We believe this to be the same tour and the difference due to a discrepancy of 1 unit in the distance of one particular link used in the tour. Our distance matrix was obtained from Ref. 4.

produced the tour with cost 12,955 which we conjecture to be the optimal solution.

Statistics collected on more than 1000 3-opt tours indicate a probability ≈ 0.02 for a 3-opt tour to be optimal.

A typical run of 100 cases with reduction in sets of 10 appears in Table III.

A few highlights in the reduction process are illustrated in Figs. 4 and 5 below.

An example of an "unfortunate" run where a link not in the optimal tour is committed in the early stages of reduction is shown in Table IV. Note that improper reduction appears in round 3 and as a consequence the subsequent values of d drop sharply. For the purpose of counting the occurrences of the optimal tour, only the first 3 rounds are considered (subsequent rounds have $d < r/2$) giving us 0 out of 30 for this run. As can be seen, we do no worse than to obtain the best tour obtained by Karg and Thompson in Ref. 4. Furthermore, the computation time involved in the first round usually exceeds 40 percent of the total computation time so that even when improper reduction happens, the total computation time is still less than that for obtaining 30 3-opt tours without reduction.

7.7 *A 105-City Problem*

To test the effectiveness of our method, a 105-city problem was constructed from the 48-city problem and 57-city problem using the facts that $u_{30,33}$ is the largest link (cost 669) in the best tour for the 48-city problem and $u_{40,46}$ (cost 685) is the largest link in the best tour for the 57-city problem. Thus, city 30 of the 48-city problem was connected to city 40 of the 57-city problem by a link with cost 10; similarly, city 33 of the 48-city problem was connected to city 46 of the 57-city problem

TABLE III — A TYPICAL 57-CITY RUN

SRR	n	r	d	C_M	C_m	$\bar{C}$	t	T
1	57	10	10	13,456	12,986	13,175.5	1	50.5
2	42	10	10	13,427	12,985	13,179.5	1	68.8
3	36	10	9	13,262	12,985	13,092.6	1	80.7
4	34	10	10	13,416	12,985	13,122.3	1	91.3
5	34	10	8	13,299	12,966	13,120.1	1	101.0
6	33	10	8	13,340	12,985	13,173.4	1	110.9
7	33	10	6	13,214	12,955	13,046.5	3	119.8
8	33	10	8	13,346	12,955	13,122.3	1	129.3
9	33	10	8	13,300	12,955	13,132.0	1	138.8
10	33	10	8	13,473	12,985	13,156.6	1	149.2

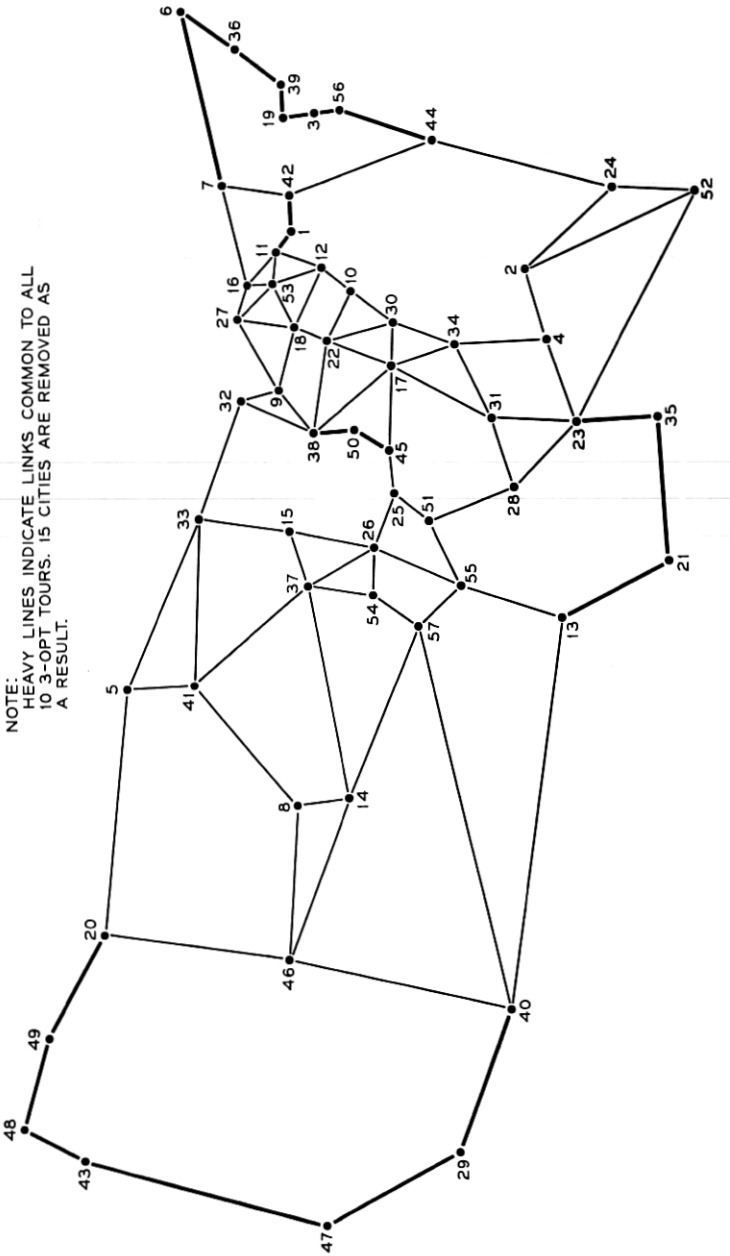

Fig. 4 — Union of all links found in the first 10 3-opt tours in the 57-city problem.

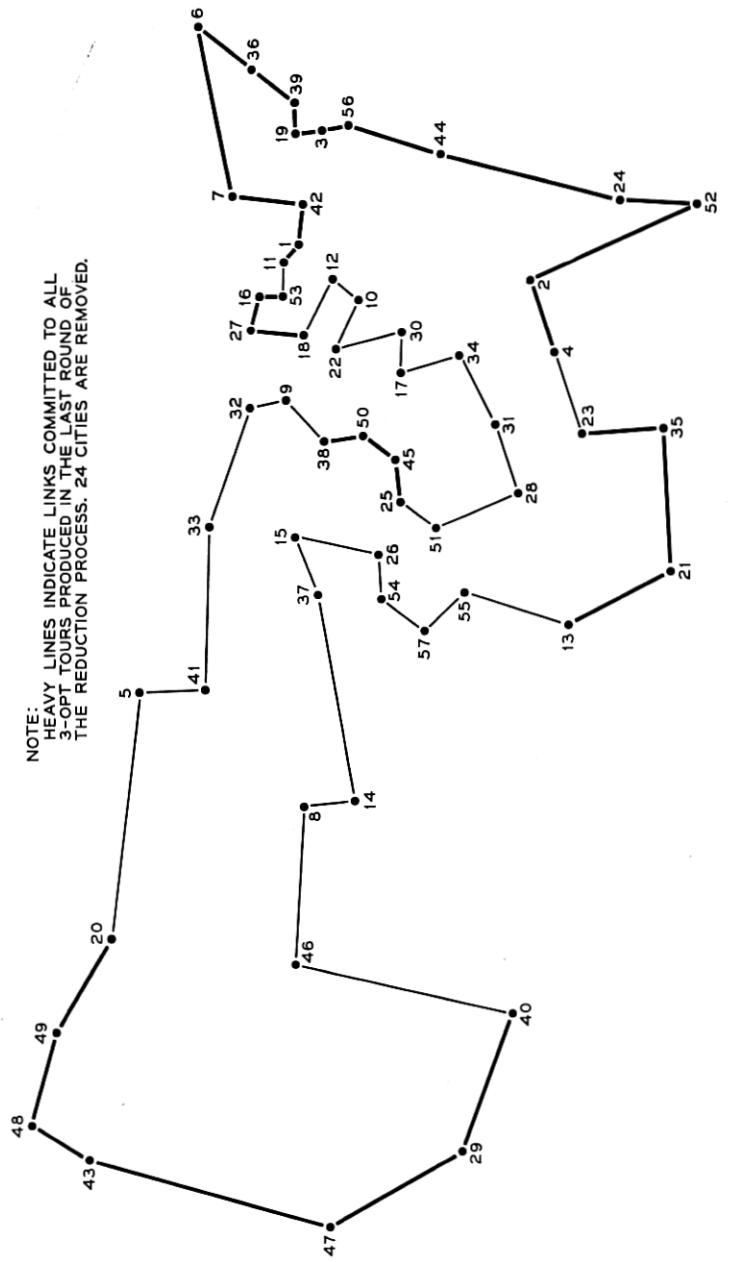

Fig. 5 — The conjectured optimum tour in the 57-city problem.

TABLE IV — A 57-CITY RUN WITH IMPROPER REDUCTION

SRR	n	r	d	C_M	C_m	$\bar{C}$	t	T
1	57	10	10	13,741	12,993	13,430.7	1	45.7
2	45	10	10	13,416	12,986	13,123.6	1	73.4
3*	37	10	9	13,197	12,997	13,091.7	1	87.2
4	30	10	4	13,114	12,985	13,001.8	3	94.2
5	23	10	4	13,012	12,985	12,991.7	2	97.4
6	22	10	3	12,997	12,985	12,990.2	2	100.2
7	21	10	2	12,986	12,985	12,985.2	8	102.7
8	17	10	2	12,986	12,985	12,985.7	3	104.2
9	17	10	2	12,986	12,985	12,985.6	4	105.9
10	17	10	2	12,986	12,985	12,985.7	3	107.2

* Improper reduction appears in this round.

by a link of cost 10. All other links between the cities in the 48-city problem and the 57-city problem were assigned random costs varying from 686 to 750, while links between the cities in the same problem remain unchanged. We purposely made those link costs moderate compared to big links in each of the two problems in order to induce sufficient mixing when a random tour is reduced to a 3-opt tour. From the method of constructing the problem there is a tour (the conjectured best tour) for this 105-city problem for which the cost is (12,955 + 11,461 + 20) − (669 + 685) or 23,082. Since the probability of obtaining the tours with costs 12,955 and 11,461 are $\approx$0.02 and 0.05, respectively, we expect that the probability of a 3-opt tour being optimal in this 105-city problem is less than 0.001. Computation time per 3-opt tour without reduction is about 35 seconds. A run of 20 3-opt tours was made with one round of reduction which reduced the number of cities from 105 to 81. Total computation time for the 20 3-opt tours was 476 seconds for an average of 23.8 seconds per local optimum. Although we did not obtain the conjectured best solution the 3-opt tour costs are surprisingly good; the worst being 24,581 and the best 23,096. An interesting fact about the 3-opt tours obtained is that besides the two short links bridging the two problems, at least one other pair of links connecting two cities in different problems appear in all the 3-opt tours obtained, indicating that this 105-city problem has a structure by itself and is not merely the conjunction of two separate problems. This is indeed what we intended it to be.

This we believe is the largest traveling salesman problem ever attempted. Reasonable estimates indicate that we may be able to solve a problem of this size with the reduction technique in 100 minutes* with a probability of success >0.5.

* See Appendix C.

7.8 *Other Experiments*

Experiments with 2-opt tours were moderately successful for smaller problems. A 20-city, 2-opt tour may be obtained in approximately 0.048 seconds with a probability of being optimum ≈ 0.06. The decrease in time compared with 3-opt tours is by a factor of 5. The decrease in probability shows that our 3-opt procedure may still be the best. When the number of cities becomes larger, the decrease in probability is so sharp as to make runs with the 2-opt procedure undesirable.

Similar experiments with programs to produce 4-opt tours show an increase of computation time per local optimum by a factor of $0.8n$,* while the chances of obtaining the optimal solution are not noticeably increased. Practically all 3-opt tours are 4-opt and the ranges of tour costs are not noticeably improved.

VIII. CONCLUSIONS AND DISCUSSION

As mentioned earlier, the methods we used here in solving the traveling salesman problem were based upon heuristic principles believed to be of general applicability to various optimization problems. These may be roughly summarized as follows:

8.1 *Probabilistic Approach vs Deterministic Approach*

In dealing with problems similar to a large traveling salesman problem, where a really efficient algorithm for the best solution is unavailable, it is in general time consuming, if not entirely hopeless, to work on refinement techniques to obtain the best solution. Rather, the approach should be to develop a technique by which good locally optimal solutions can be obtained very fast, and with reasonable probability that among the locally optimal solutions, we may indeed find the best. (In actual applications, the best of a set of good locally optimal solutions, even though it may not be the best, will be close enough to the best solution as to offer a satisfactory answer for most problems.) The fact that we generate for a given traveling salesman problem a large number of 3-opt tours rather than develop means of further improving a 3-opt tour is based on this principle. The fast computation gives us the ability to generate many locally optimal solutions which, coupled with a reasonable probability that a locally optimal solution is optimal, guarantees us a very high probability of success for solving the problem.

* Although the ratio of $\binom{n}{4}$ and $\binom{n}{3}$ is $\frac{n-3}{4}$, for each 4 links removed, there are 48 ways of putting the 4 strings together, compared with 8 ways of putting 3 strings together.

8.2 *Choice of Algorithm*

Consider two algorithms A_1 and A_2 which will produce locally optimal solutions for a given problem in computation times t_1 and t_2. Suppose A_1 is "stronger" then A_2 in the sense that A_1 produces locally optimal solutions which are optimal more frequently than A_2, say A_1 with probability p_1 and A_2 with probability p_2 and $p_1 > p_2$. A_1 need not be preferred to A_2 if t_1 is disproportionally greater than t_2. This can be seen as follows: for a given problem, suppose we are permitted a total computation time t, (which may be the amount of computing time we are able to buy with available funds), so that in time t, we can perform $[t/t_i] = k_i$ experiments with algorithm A_i. Then the probability that among the k_i locally optimal solutions obtained, we indeed have the optimal solution is $p_i^* = 1 - (1 - p_i)^{k_i}$. This is the quantity we should maximize. In the event that we are interested in only good approximate solutions, we should choose an algorithm A_1 over another algorithm A_2 such that for given amount of time t, the best of the k_1 locally optimal solutions relative to A_1 is better than the best of the k_2 locally optimal solutions relative to A_2.

As an example, consider the sequence of algorithms A_λ to produce λ-opt tours with associated probabilities p_λ and computing times t_λ. For the range of the size of problem we are dealing with, say from 10 to 100 cities, we have reason to believe that p_3^* is the largest among the p_λ^*'s, as indicated by our computation results, and that the best tour produced from k 3-opt tours is at least as good if not better than the best λ-opt tours produced in comparable time using any other A_λ.

8.3 *Random Improvement vs Steepest Descent*

Within the algorithm for obtaining a locally optimal solution, substantial saving in time can be achieved by not attempting to find the best improvement possible at any stage, but rather to take the first improvement that occurs. In general, the method of steepest descent tends to increase the computation time disproportionally and should not be used. Attention should be directed to finding improvements with a minimum amount of computation rather than to making the maximum improvement possible at each step.

8.4 *Restricting Search in Increasingly Smaller Domain*

When sufficient information has been gathered about the problem, ways and means should be investigated to restrict substantially the domain of search. This should be done even with the possibility that

the optimal solution may be lost in the process (of course only with small probability). This is well illustrated by our reduction procedure described in Section VII.

IX. ACKNOWLEDGMENT

The author would like to thank T. H. Crowley and A. J. Goldstein for many stimulating ideas which materially helped in producing these results.

APPENDIX A

k-Length String Optimization

In the first algorithm, called k-length string optimization, a dynamic programming technique is used to optimize tours (or strings) of k cities for $k \leqq 13$.* The algorithm is a slight modification of that given in Ref. 2 by Held and Karp. However, we achieve a significant reduction in computation time by taking advantage of the fact that the distance matrix is symmetric.

Let the k cities be represented by the integers 1 through k, $X = \{2, 3, \cdots, k\}$ be the set of integers from 2 to k and S be a subset of X consisting of m elements; i.e., $|S| = m$. Let $C(S,i)$ with $i \in S$ denote the minimum cost of starting from city 1 and visiting all cities j in S, terminating at city i. Then the quantities $C(S,i)$ can be computed recursively as follows:

$$C(\{i\},i) = d_{1i} \tag{1}$$

$$C(S,i) = \min_{j \in S-i} [C(S - i,j) + d_{ji}]. \tag{2}$$

For example, if $S = \{3,5,9,11\}$, then $C(S,9)$ = cost of best string from 1 to 9 through 3, 5, 11

$$= \min \begin{cases} C(S - 9,3) + d_{3,9} \\ C(S - 9,5) + d_{5,9} \\ C(S - 9,11) + d_{11,9}. \end{cases}$$

We note here that $S - i$ is a subset of X such that $|S - i| = |S| - 1$, and thus the quantities $C(S - i,j)$ have all been computed a step before in the recursion scheme.

For $k = 2t + 1$, we recursively compute and store $C(S,i)$ for all

* Storage requirements in dynamic programming seriously limit the size of the problem that can be handled.

subsets S of X for $|S| = 1$ up to $|S| = t$. Then we successively compute $C(S,i)$ for $|S| = t + 1$. At this point, if we denote the complement of these S's in X by $\bar{S}$, we see that $S^* = \bar{S} \cup \{i\}$ is a subset of t elements and $C(S^*,i)$ has already been computed. The cost r of the optimal tour T is therefore given by

$$r = \min_{S} [\min_{i \in S} [C(S,i) + C(S^*,i)]]$$

where S ranges over all subsets of X containing $t + 1$ elements. Since either S or S^* must contain say city 2, we may further restrict the range of S to only those containing the city 2.

For $k = 2t$ the procedure is similar except that we need not compute $C(S,i)$ for $|S| > t$.

The order of the cities in the tour T can now be determined. With the "middle" city i and sets S,S^* determined from the expression for r, we find a city i_1 in $S - i$ such that $C(S - i,i_1) + d_{ii_1} = C(S,i)$; then a city i_2 in $S - \{i,i_1\}$ such that $C(S - \{i,i_1\},i_2) + d_{i_1 i_2} = C(S - i,i_1)$ and so on until S is exhausted, and similarly for the set S^* to produce the tour $T = 1, \cdots, i_2, i_1, i, i_1^*, i_2^*, \cdots, 1$.

This algorithm can be used, with a slightly modified distance matrix, to find the minimum string from city 1 to city k through the cities $2, 3, \cdots, k - 1$ and its associated cost $C(X,k)$. We set $d_{1k} = C(\{k\},k) = 0$ and $C(S,k) = \infty$ for $|S| \geqq 2$ in the recursive computation scheme described above. This insures that city k is next to city 1 in the tour produced, and hence by removing the link from city 1 to city k we get the best string with 1 as our initial city and k the terminal city.

Given a n-city problem, a tour T through the n cities is said to be locally optimal relative to the k-length string optimization algorithm if every ordered set of k consecutive cities in T, say $(i_{\alpha+1}, \cdots, i_{\alpha+k})$, subscripts reduced modulo n, is optimal as a string from $i_{\alpha+1}$ to $i_{\alpha+k}$ going through the cities $i_{\alpha+2}, i_{\alpha+3}, \cdots, i_{\alpha+k-1}$. The above program may be used to produce locally optimal tours of this type for any n-city problem without change as follows: Consider a random initial tour represented by the permutation $P = (i_1, i_2, \cdots, i_n)$. We map the first k cities $i_1, i_2, \cdots, i_k$ onto $1, 2, \cdots, k$ and use the program to find the best string from 1 to k, say $1, j_2, j_3, \cdots, k$. The associated permutation $P^* = (i_1, i_{j_2}, i_{j_3}, \cdots, i_k, \cdots, i_n)$ gives us a tour whose cost is no larger than P. When there is no gain in cost from P to P^* it means that $i_1, i_2, \cdots, i_k$ is already optimal as a string. Next, we rotate P^* by a length δ (relatively prime to n) and repeat the process until there are n consecutive rotations without a decrease in cost. Then every ordered

set of k consecutive cities in the final permutation is now optimal as a string. Experiments with this procedure indicated, however, that it is time consuming and not nearly as effective as the second algorithm which we have discussed in the paper.

APPENDIX B

Outline of Computer Program to Produce 3-Opt Tours from Random Initial Tours

Notation:

n	number of cities
(d_{ij})	distance matrix
r	number of 3-opt tours desired before reduction
u_{ij}	link connecting city i and city j
t_i	ith city in the tour
S	the union of the set of links found in previously generated 3-opt tours
q	a program branching parameter.

1. $S = \phi$
2. Do through (14), $m = 1, 1, r$.
3. Generate a random tour $(t_1, t_2, \cdots, t_n)$.
4. $q = 0$ if $m = 1$, otherwise $q = 1$.
5. Do through (12), count $= 1, 1, n$.
6. If $q = 0$, skip 7.
7. If $u_{t_1 t_n} \varepsilon S$ go to 12 (special feature).
8. Do through (11), $k = 1, 1, n - 3$.
9. Do through (11), $j = k + 1, 1, n - 1$.
10. If $d_{t_k t_{j+1}} + d_{t_1 t_j} \leqq d_{t_1 t_{j+1}} + d_{t_k t_j}$ set $d = d_{t_k t_{j+1}} + d_{t_1 t_j}$ and $\alpha = 16$, otherwise set $d = d_{t_1 t_{j+1}} + d_{t_k t_j}$ and $\alpha = 18$.
11. If $d + d_{t_{k+1}, t_n}$ (cost of links added) $< d_{t_1 t_n} + d_{t_k t_{k+1}} + d_{t_j t_{j+1}}$ (cost of links removed) go to α (otherwise loop).
12. $(t_1, t_2, \cdots, t_n) = (t_n, t_1, \cdots, t_{n+1})$ (rotate).
13. If $q = 1$, set $q = 0$ and go to 5 (almost 3-opt tour obtained).
14. $S = S \cup$ links in $(t_1, t_2, \cdots, t_n)$ (3-opt tour obtained).
15. Go to reduction (see description in Section VI).
16. $(t_1, t_2, \cdots, t_n) = (t_{j+2}, \cdots, t_n, t_{k+1}, \cdots, t_j, t_1, \cdots, t_k, t_{j+1})$ (links exchanged and tour perturbed).
17. Go to 5 (treat improved tour as initial tour).
18. $(t_1, t_2, \cdots, t_n) = (t_{j+2}, \cdots, t_n, t_{k+1}, \cdots, t_j, t_k, \cdots, t_1, t_{j+1})$.
19. Go to 5.

APPENDIX C

Estimates on the Probability that a 3-Opt Tour is Optimal

From the statistics collected on running many problems, we estimate that a 3-opt tour in a 10-city problem of some difficulty has a probability of 0.5 of being optimal, and in general, this probability seems to decrease by a factor of 2 for each addition of 10 cities. We shall denote these estimates by $p(n)$ and use them as basis for our calculations. We have

$$p(n) \approx 2^{-n/10}.$$

So far, these estimates has been close for problems in the range between 30 and 60 cities and is too conservative for smaller problems, or exceptionally easy problems like the 33-city problem. For exceptionally difficult problems, we define $p^*(n) = \frac{1}{4}p(n)$ as the estimate for the "worst." Computation time $t(n)$ per 3-opt tour in an n-city problem averages $30n^3$ microseconds without reduction. With reduction, we can obtain 100 3-opt tours usually in the amount of time needed for 25 3-opt tours without reduction.

A few examples will show how to estimate time needed to "solve" a given problem. We consider a problem solved if the probability p of obtaining the optimal solution is $\geqq 0.99$. It should be noted that the estimates are heuristic in nature and tend to be on the conservative side.

Example 1 — Given a 20-city problem, we have $p(20) = \frac{1}{4}$, $t(20) = 0.24$ second. In order to have

$$(\tfrac{3}{4})^k \leqq 0.01$$

we must have $k > 16$. Thus 4 seconds of computation should be adequate. In the "worst" case $p^*(20) = \frac{1}{16}$, $k = 72$ should be adequate. Computation time is about 18 seconds if reduction is not used and about 5 seconds if reduction is used.

Example 2 — For a 40-city problem we have $p(40) = \frac{1}{16}$, $t(20) \approx 2$ seconds. With $k = 72$, we need 144 seconds without reduction and about 40 seconds with reduction. In the "worst" case $p^*(40) = \frac{1}{64}$ and $k = 300$ is sufficient. Running the program in 3 independent runs of 100 with reduction, total computation time needed is about 2.5 minutes.

Example 3 — For a 60-city problem, we have $p(60) = \frac{1}{64}$, $t(60) \approx 6.5$ seconds. With $k = 300$ and using reduction, about 8 minutes of computa-

tion time is required to guarantee $p > 0.99$. In the "worst" case with $p^*(60) = \frac{1}{250}$, the same computation will yield $p \approx 0.7$.

Example 4 — For a 100-city problem $p(100) = \frac{1}{1024}$ and $t(100) \approx 30$ seconds. With $k = 800$ we have $p \approx 0.54$. With reduction, this can be achieved in about 100 minutes.

APPENDIX D

Two Conjectures

Using the notation of Section III, the following appears to be an interesting problem: Find the minimum number k for which $C_k = C_{k+1} = \cdots = C_n$. For fairly large k it appears, at least intuitively, that a k-optimal tour should be optimal, since by Theorem 1 (Section III), if it is not optimal, then its index can be at most $n - k - 1$. Due to the intrinsic difficulties in this problem we can only state the following conjectures:

Conjecture 1 — $C_{n-1} = C_n$. That is, any tour which is $(n - 1)$-optimal is also optimal.

Conjecture 2 — $C_k = C_{k+1} \rightarrow C_k = C_{k+1} = \cdots = C_n$.

Conjecture 1 can be verified for $n \leqq 6$. Also an $(n - 1)$-optimal tour which is not optimal must have index 0, hence all of its links are inadmissible. Furthermore, it must also be n-length string optimal. The existence of such a tour seems to be extremely unlikely. Work on Conjecture 1 has led to the following interesting problem in graph theory, which is equivalent to Conjecture 1, and yet involves no concept of any distance matrix.

Problem: Suppose we are given a graph with n vertices and $2n$ links which can be partitioned into 2 sets of n links, each of which form a Hamiltonian circuit. Does there exist another partition with the same property?

If the answer to the above problem is always in the affirmative, then we can prove Conjecture 1 in the following way. Consider a graph consisting of an optimal tour T and an $(n - 1)$-optimal tour T^* which is not optimal. Since T^* has index 0, T, and T^* have no link in common and thus serve as a partition into 2 sets of n links, each of which form a Hamiltonian circuit. Let the other partition with the same property be tours T_1 and T_2. Since T_1 and T_2 uses the same set of $2n$ links of T and T^*,

$$C(T_1) + C(T_2) = C(T) + C(T^*) < 2C(T^*).$$

Hence one of the tours, say T_1 must have cost $C(T_1) < C(T^*)$. But T_1 and T^* must have at least one link in common; hence T^* cannot be $(n-1)$-optimal.

On the other hand, suppose there is a graph with n vertices and $2n$ links which can be partitioned into 2 sets of n links A and B, each of which form a Hamiltonian circuit and that no other partition with the same property is possible. Let the n vertices represent n cities. We construct a distance matrix as follows: let each link in A be assigned a distance d_a ; each of the $n-1$ links of B be assigned a distance d_b and the remaining link in B, d. Let all other links in the complete graph of n nodes have distance $> \max [n \cdot d_a , (n-1)d_b + d]$. Suppose $nd_a < (n-1)d_b + d$. Then it is clear that the set of n links in A form the optimal tour while the set of n links in B form an $(n-1)$-optimal tour. Furthermore, we can make $d_a > d_b$ so that the $(n-1)$-optimal tour which is not optimal contains $n-1$ smallest links, and by making d large, we can also make the "next best" tour as poor as possible compared with the optimal tour.

Conjecture 2 is obviously true for $k = n-1$. We prove it is true for $k = 1$. By hypothesis, $C_1 = C_2$, that is, no tour crosses itself. Suppose there is a tour T which is not optimum, then we may transform T into the optimal solution by a sequence of transpositions of immediate neighbors, each step being equivalent to an inversion. Since the cost must finally reduce to the cost of the optimal tour, at some point we must have a tour with the property that transposing 2 immediate neighbors reduces the cost, giving us a crossover situation contradicting the hypothesis.

REFERENCES

1. Dantzig, G. B., Fulkerson, D. R., and Johnson, S. M., Solution of a Large-Scale Traveling Salesman Problem, Oper. Res., *2*, 1954, pp. 393–410.
2. Held, M. and Karp, R. M., A Dynamic Programming Approach to Sequencing Problems, J. Soc. Ind. Appl. Math., *10*, No. 1, March, 1962, pp. 196–210.
3. Croes, G. A., A Method for Solving Traveling Salesman Problems, Oper. Res., *5*,
4. Karg, R. L. and Thompson, G. L., A Heuristic Approach to Solving Traveling Salesman Problems, Manage. Sci., *10*, No. 2, January, 1964, pp. 225–247.
5. Sherman, G. and Reiter, S., Discrete Optimizing, Inst. Quant. Res. Econ. and Manag., *37*, Purdue University, 1963.
6. Flood, M M., The Traveling Salesman Problem, Oper. Res., *4*, 1956, pp. 61–75.