[Reprinted from Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems, held on March 16-18, 1964, at the Thomas J. Watson Research Center, Yorktown Heights, N. Y.]

8
The Traveling Salesman Problem

R. E. Gomory
IBM *Corporation*

The traveling salesman of the traveling salesman problem is interested in only one thing—money. He sets out to pass through a number of points, usually called cities, and then returns to his starting point. When he goes from the ith city to the jth city, he incurs a cost $c_{i,j}$. His problem is to find that tour of all the points (cities) that minimizes the total cost.

Now this problem, although easy to state, turns out to be an extremely difficult one to solve. This is surprising, because the traveling salesman problem is almost indistinguishable from a very easy problem—the assignment problem.

The assignment problem is illustrated in Figure 1. That figure shows a situation in which there are n nodes, or men, which can be assigned to n other nodes, or jobs. If the ith man is assigned to the jth job, there is a cost for that assignment $c_{i,j}$. The problem here is simply to find a permutation, or assignment, which assigns to every job a man and minimizes the total cost of the assignment. One assignment or permutation, ϕ, is illustrated in the top of the figure.

Another way of illustrating the assignment problem, which makes the connection with the traveling salesman problem more obvious, is not to have two sets of nodes representing jobs but only one set. When the ith man is assigned to the jth job, an arrow is drawn from the ith node to the jth node, and the cost $c_{i,j}$ is incurred. The problem is, of course, unchanged. It is to find a permutation ϕ which assigns to each node i a successor $\phi(i)$ such that the total cost is minimized. Now, as the lower part of Figure 1 shows, the only difference between the assignment problem and the traveling salesman problem is that we allow small closed loops in the assignment problem. The traveling salesman problem, as can

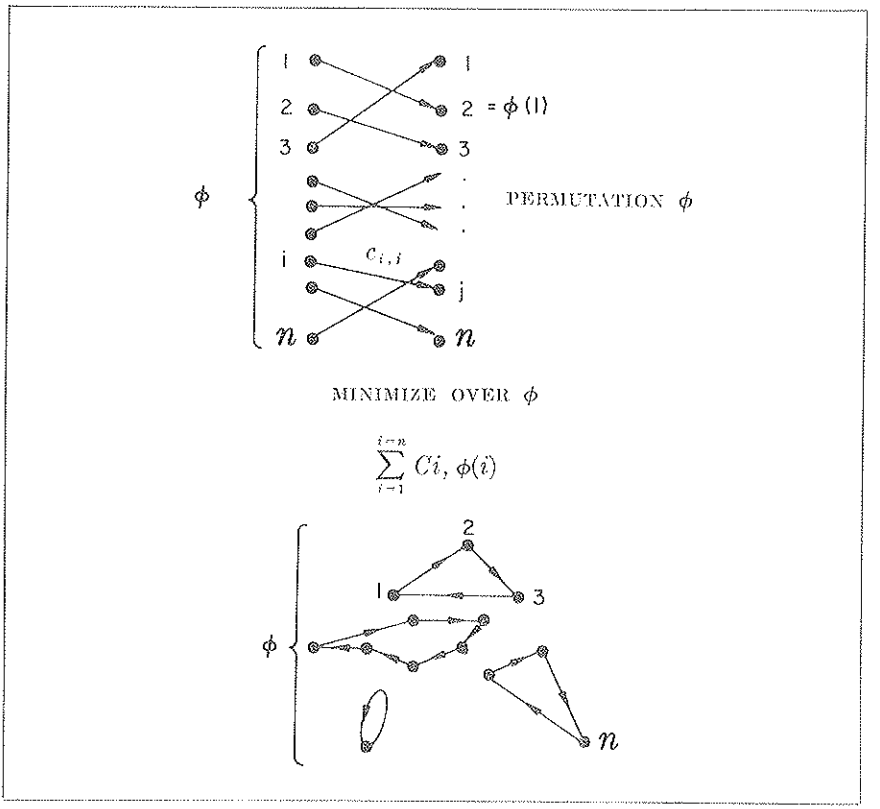


FIGURE 1. Assignment problem

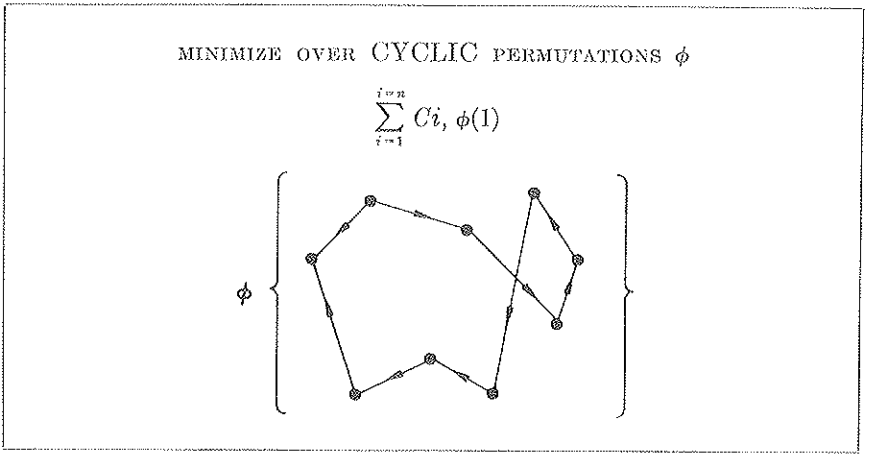


FIGURE 2. Traveling salesman problem

be seen from Figure 2, is almost exactly the same. We are looking for a permutation ϕ that gives to every node i a successor j and which minimizes the total cost. In the traveling salesman problem, we must have a cyclic permutation; the diagram must not break up into little loops.

The two problems, the assignment problem and the traveling salesman problem, may not seem to be very different; but they are, and we cannot really ignore either of them. They both have a considerable number of applications. I will not try to list these applications in general. However, for the traveling salesman problem, there is one very important application that I will outline. This is the problem of sequencing jobs on a machine. Let us assume that there are a number of jobs to be run in succession on the machine. After the ith job is run, the machine is set up to run the jth job, and a certain cost $c_{i,j}$ is incurred by this setup. The problem is to run the jobs on the machine one after the other so that the total changeover or setup cost is minimized. Now this is very close to the traveling salesman problem because one must run through all of the jobs, incurring a cost $c_{i,j}$ going from i to j. If the jobs are represented by points, this is the same as asking for a path that runs through all these points (jobs) with the least total cost. One difference is that we are not now asking to come back to the starting point. Actually, this is of very little consequence, and there are tricks that transform the sequencing problem, just described, into the traveling salesman problem. The two problems are essentially the same, and the traveling salesman problem is easier to work with, since one deals with complete permutations.

If we take these two problems, the assignment problem and the traveling salesman problem, which appear to be only slightly different, and if we ask how much work must be done to obtain the optimal permutation, the answers are tremendously different in the two cases. In the assignment problem, the optimal assignment can at present be obtained in $O(n^4)$ elementary steps. The corresponding bound in the traveling salesman problem is $O(n^2 2^n)$ elementary steps. So these two problems which seem to be so close are, at least at the present time and so far as our bounds are concerned, radically different.

One of the ways that this difference could be investigated is to take a look at the polyhedra associated with these problems to see if they look more or less alike. This is a reasonable procedure because there has been considerable success on the assignment problem by using vertex-to-vertex searches on polyhedra—this is in fact the linear programming approach. Since we have to specify which polyhedron is under discussion, let us first take the assignment problem. Any permutation ϕ provides a feasible solution to it, and any permutation ϕ can be represented as an $n \times n$ matrix with a 1 in the ij position if i goes to j and with a 0 otherwise. Thus, with every permutation there is an associated $n \times n$ matrix,

which can then be regarded as a point in n^2-dimensional space. Taking the convex hull of these points gives a certain polyhedron. For the assignment problem, the associated polyhedron is the convex hull of all permutations. For the traveling salesman problem, it is the convex hull of those permutations that are cyclic. In any case, to solve our problem, we try to maximize a linear form $\sum c_{i,j} x_{i,j}$ over these polyhedra.

A number of facts about these polyhedra were assembled by Heller (1954 and 1955), by Motzkin (1956), and by Kuhn (1955), and are illustrated in Figure 3. First, what about the dimension of the polyhedron? Although these points are in n^2-dimensional space, they could perfectly well all be in a lower-dimensional subspace; in fact, they are. The dimension of the polyhedron P of permutations is $(n-1)^2$, and that of the polyhedron Q of cyclic permutations is $(n-1)^2 - n$. So the convex of all tours is a little bit smaller in dimension, which suggests, rather misleadingly, that this problem might be the easier one. Something further, that is perhaps more relevant to a vertex-to-vertex search, is the question of the neighborliness of the points on the polyhedra. Should one expect to travel a long distance to get from a starting vertex to the optimal one in doing a vertex-to-vertex search? To answer this, some measure of the neighborliness of points on the polyhedra is needed.

Two measurements of neighborliness will be discussed here.

One way to measure neighborliness is to say that two points are very close if they have a one-dimensional face in common. That is about as neighborly as vertices can be on a polyhedron. Let us also say that they are fairly near if they have a two-dimensional face on which both vertices lie, and so on up. In fact, we will say that points are n-neighbors if they both lie on the same n-dimensional face. According to this index of neighborliness, all vertices on these polyhedra are very close to each other because the most unneighborly pair of points are neighbors of order $[n/2]$; in other words, there is a face of dimension $[n/2]$, or lower, containing any pair of points on the polyhedra. This is a very low dimensional face compared with the dimension of the space in which the polyhedron is, and this is just as true for Q, the convex of tours, as it is for the polyhedron P. In fact, it is slightly more so for Q, since it is known that on Q, for $n \leq 5$, each vertex is a one-dimensional neighbor of every other vertex; it is only for $n = 6$ that it takes two steps to get from one point to another.

This suggests another measure of neighborliness, which is the number of edges to be traversed to get from one vertex to another. In the case of P, this number is known, because two vertices are one-step neighbors if their difference, the difference permutation, contains only one cycle. The number of steps to get from one vertex to another is the number of cycles in the corresponding difference permutation. Therefore, the maximum distance on P in this sense is again an integer part of $n/2$,

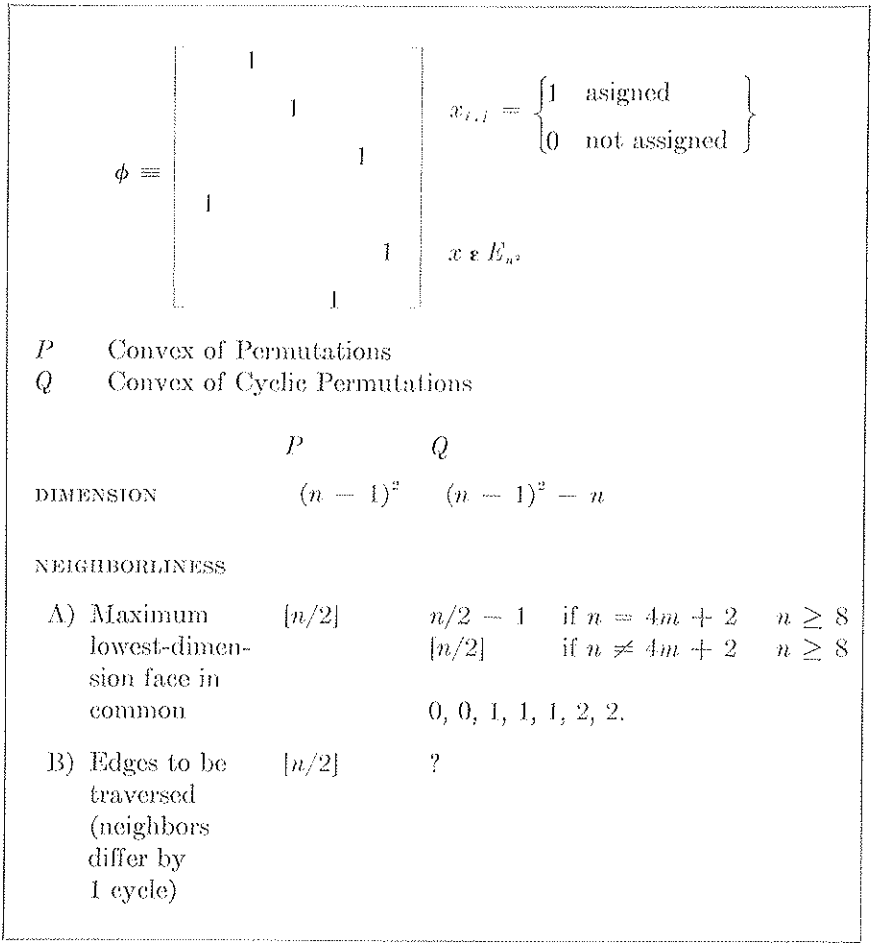


FIGURE 3. Polyhedra

namely, ($[n/2]$); so this measure of neighborliness comes out the same as the other measure. For Q, however, no facts about this type of distance or neighborliness are known, and this is the first place that the two polyhedra seem to look different, but, of course, this difference is not certain.

We now come to another way of looking at the polyhedra in which they definitely do look different; this is again suggested by a linear programming approach. For the linear programming approach, the number of faces is relevant, as each one of these represents an inequality of the linear programming method. So it is relevant to ask: If we had to write down all the equations describing these polyhedra for a linear programming approach, how many would have to be written?

Let us first consider the assignment problem. The permutations of the assignment problem satisfy the two sets of equations shown in Figure 4. These are the equations that cut the dimension of the polyhedron P down from n^2. In addition, in this lower-dimensional face, there are the nonnegativity conditions, also shown in Figure 4, which actually provide the faces. There are $(n-1)^2$ of these. This then is the number of faces. (One can show that these inequalities each give faces; none is redundant or misses the polyhedron entirely.) When we consider the convex Q, the story is very different. First of all, there are the sets of equations which cut down the dimension. There are a few more of these than there are

ASSIGNMENT PROBLEM

$$x_{i,j} \geq 0 \qquad \left\{ \begin{array}{l} \sum_i x_{i,j} = 1 \\ \sum_j x_{i,j} = 1 \end{array} \right\} \quad \begin{array}{l} j = 1, \cdots, n \\ i = 1, \cdots, n \end{array}$$

TRAVELING SALESMAN

$n = 5$

I (20) $x_{i,i} \geq 0$ $\quad (i, j)$

II (10) $x_{i,j} + x_{j,i} \leq 1 \quad i \neq j$

$$\text{I, II} \left\{ \begin{array}{l} \sum_i x_{i,j} = 1 \\ \sum_j x_{i,j} = 1 \\ x_{i,i} = 0 \end{array} \right\} \quad \begin{array}{l} j = 1, \cdots, n \\ i = 1, \cdots, n \end{array}$$

III
$$\left\{ \begin{array}{ll} (60)\ x_{i,j} + x_{j,i} + x_{r,s} - x_{s,t} - x_{t,r} \leq 1 & i, j, r, s, t \text{ distinct} \\ (120)\ 2(x_{i,j} + x_{j,i}) - x_{i,r} + x_{j,r} - x_{s,j} + x_{s,i} - x_{r,s} \leq 2 & i, j, r, s \text{ distinct} \\ (60)\ x_{i,j} + x_{j,i} + x_{i,r} + x_{r,j} + x_{s,j} + x_{r,s} + x_{j,t} + x_{t,r} \geq 1 & i, j, r, s, t \text{ distinct} \end{array} \right.$$

In general, faces *include* three classes:

I $n(n-1), \quad x_{i,j} \geq 0$

II $P_1 \geq 2^{n-1} - n - 1$ (elements of $k \times k$ principal

III $P_2 \geq \dfrac{(n-1)!\,n!}{2^n}$ minor $= k - 1, 2 \leq k \leq [n/2]$)

FIGURE 4. Faces

in the case of the assignment problem. When we come to the inequalities, we have the nonnegativity conditions, just as in the assignment problem, and then another group (group II in Figure 4), which says that we cannot have an assignment both from i to j and from j to i, because then we would have a two-step loop and not a tour. In Figure 4 these inequalities are illustrated for the case $n = 5$, where there are ten of them. There is a third set (group III), which contributes 240 more inequalities in the case $n = 5$; the origin of this group is too complicated to be given here. There is also a fourth set of 120 more inequalities not shown in Figure 4.[1] Together, the four groups give 390 faces in contrast with the 20 faces of the assignment problem, so we start to see a difference. The assignment problem yields a very simple polyhedron; the traveling salesman problem yields a multifaceted one. As Figure 4 shows, the traveling salesman problem in general yields a polyhedron whose faces from class II are almost 2^{n-1} in number, while in class III they are even more numerous. Those of class II that extend the notion of eliminating two-step tours, which appeared in the case $n = 5$, are the inequalities that eliminate subtours or lengths less than $[n/2]$.

This very large number of faces would discourage most people from trying to use the linear programming approach on the traveling salesman problem, but it did not discourage Dantzig, Fulkerson, and Johnson, who tried it anyway. They were, perhaps, a little encouraged by the fact that the number of faces does not seem to be quite so excessively large in the symmetric case ($c_{i,j} = c_{j,i}$). In the symmetric problem, groups I and II of the inequalities (Figure 4) are sufficient for $n = 5$, but not for $n = 6$. For $n = 7$, there are already 2,177 inequalities. So the symmetric situation, although better, is not good. However, Dantzig, Fulkerson, and Johnson worked on a traveling salesman problem involving 42 cities and obtained the solution shown in Figure 5. This was quite a remarkable achievement and I would like to indicate, roughly, what they did. They would start with some tour x and a starting convex made up of a subset of the inequalities. (For instance, one could take just the nonnegativity conditions.) Call this the convex C_1. Our tour x is an extreme point of C_1. Use the simplex method to move to an adjacent extreme point e in C_1 which gives a better value. If e is a tour, repeat from it. If it is not a tour, then among the unwritten inequalities there must exist a hyperplane separating the starting point from e. Find one of these which passes through x, and add it to the problem, making a new convex C_2. Start with x again, and repeat the procedure until a tour x^* and a convex C_m are found over

---

[1] This group was overlooked in the original presentation of my paper. I would like to thank H. W. Kuhn for bringing to my attention his paper which corrects earlier work on this topic (see Kuhn, 1955).

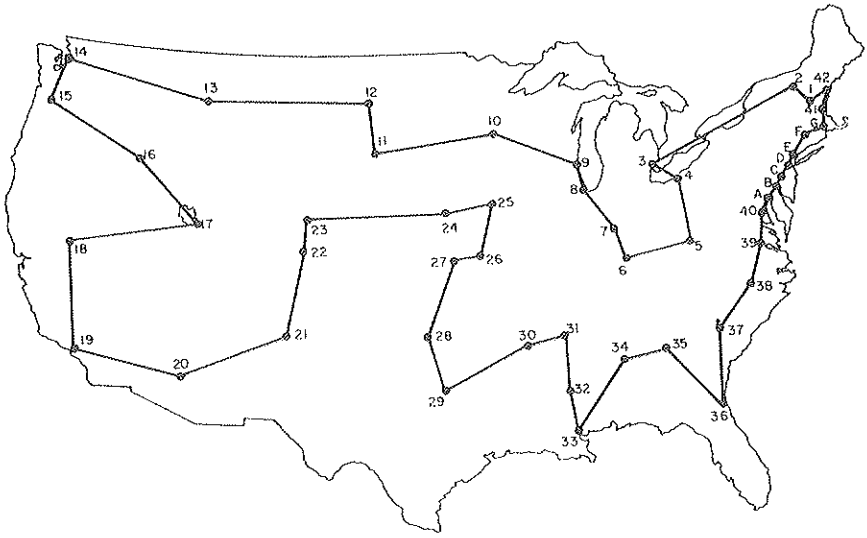


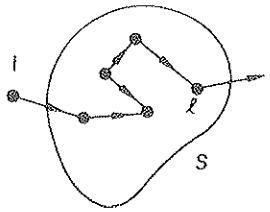
FIGURE 5

which x^* maximizes the linear form $\sum c_{i,j} x_{i,j}$ according to the usual simplex criteria.

Using this method, Dantzig, Fulkerson, and Johnson managed to solve the problem. They found that they encountered very few inequalities from the third or complicated family. Apparently, in practice, the relatively easy types, one and two, are predominant. The method they used has a very large artistic component; the way that the inequalities were produced to cut off the non-tour points was not routine but had to be invented at each step. Also, the procedure was not certain of being terminated, and there was also the problem of maintaining a basis in a compact way that I would describe as artistic rather than algorithmic. Nevertheless, they solved a traveling salesman problem of a size comparable to the best that can be solved today.

I do not see why this particular approach stopped where it did. It should be possible to use the same approach today, but in an algorithmic manner. We no longer have to be artistic about generating the separating hyperplanes or cuts, since this is now done automatically in integer programming. It seems likely that one can get over the difficulties of maintaining the basis as well. So it should be possible to do the whole thing now systematically. This is an approach one might not expect to work, but we already know that it does.

Now we will turn to an entirely different approach, one based on dynamic programming. The approach I am going to describe was invented by Bellman (1960) and, later, independently by Held and Karp (1962). Their procedure is illustrated in Figure 6.



$S \subset \{2, 3, \cdots, n\}$

$l \in S$

$C(S, l)$—Least cost for traversing S ending with l.

$$\begin{cases} |S| = 1 & C(\{l\}, l) = a_{1,l} \\ |S| = 2, \cdots, n-2 & C(S, l) = \min_{m \in S - l} \{C(S - l, m) + a_{m,l}\} \\ |S| = n - 1 & C = \min_{l \in (2,3,\cdots,n)} \{C(\{2, 3, \cdots, n\}, l) + a_{l,1}\} \end{cases}$$

FIGURE 6. Dynamic programming

Let us introduce a function $C(S, l)$. This is defined as the least cost associated with any path that starts from node 1, traverses all the elements of the set S, and ends with the element l belonging to S. The function $C(S, l)$ is easy enough to calculate if S consists of a single element; then as Figure 6 shows, $C(\{l\}, l)$ is simply $a_{1,l}$. Now let us suppose that we have $C(S, l)$ for all sets l containing some number of elements p, where p is greater than one and less than $n - 2$. Then we can find the $C(S, l)$ for a set containing $p + 1$ elements by a second recursion. For the cost $C(S, l)$ is the minimum over all m of the cost of the path that starts out at node 1, traverses all the elements of $S - l$, and ends up at m, plus the cost of then getting from node m to node l. Finally, for sets of $n - 1$ elements, there is a similar recursion which allows the computation of the cost of a complete tour. So it is clear that the various $C(S, l)$ can be computed successively, starting with the one-element sets, going to the two-element sets, and so forth, until finally a cost for the entire tour is obtained.

The main question is how much arithmetic will have to be done to get this answer.

There are $\binom{n-1}{k}$ ways to pick a subset of k elements, and in every such subset there are k elements that can serve as the last element; therefore, the minimization in the second recursion will have to be done $k\binom{n-1}{k}$ times, and the minimization can be regarded as having about $k-1$ moves, a move being an add or a compare. If this is then summed over the various-sized subsets, we get the formula

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

so the computation is $O(n^2 2^{n-3})$. All that this work gives us is the cost of the minimal tour, and there would be a certain amount of extra computation involved in going back to find out which tour gave that minimal value. However, this is negligible by comparison with the rest of the work. As far as storage is concerned, we can assume one location for each of the $C(S, l)$ values, which gives us the formula shown below and something of the order of $n2^{n-2}$ locations for the n-city problem.

Storage one location for each $C(S, l)$

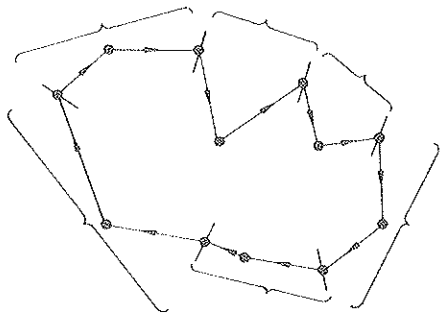
$$\sum_{k=2}^{k=n-1} k\binom{n-1}{k} = (n-1)2^{n-2}$$

IBM 7090 — 13 cities — 17 seconds

Roughly speaking then, what the dynamic programming does is to replace a roughly $n!$ problem with one of the order of 2^n. On a computer one can get up to about 13 cities without too much difficulty. At that point we begin to get a little short on storage, and times start to go up by a factor of 2 each time, roughly speaking.

Now Held and Karp did not stop at 13 cities. They took the same technique and applied it to bigger problems to obtain approximate solutions. The sort of thing they did is illustrated in Figure 7. They supposed that they had some sort of a starting solution—good, bad, or indifferent—which they would break up into little sequences of a few cities each. Then they would regard these pieces themselves as being cities. The cost of going from this piece to some other piece is the cost of going from the last node in the first piece to the first node of the second piece. This then gives the $c_{i,j}$ for a new problem.

This new problem is, of course, smaller than the original and, in fact, is usually broken up into a 13-city problem, because that can be done with reasonable rapidity by the dynamic programming technique. If one particular way of cutting up does not work, in the sense of producing



"Global" [1, 2, 3] [4, 5, 6] [7, 8, 9] ... [37, 38, 39]
[10, 11, 12] [35, 36, 37] ... [7, 8, 9]

"Local" [1] [2] [3] [4] ... [12] [13, ... , 39]
[3] [4] [1] [2] ... [12] [13, ... , 39]

FIGURE 7. Iterating the procedure

a better tour, there are other methods of cutting up that can be tried. Held and Karp called the procedure of cutting up into approximately 13 equal pieces the "global" approach:

$$[1, 2, 3]\ [4, 5, 6]\ [7, 8, 9] \cdots [37, 38, 39]$$

$$[10, 11, 12]\ [35, 36, 37] \cdots [7, 8, 9].$$

They have also a "local" approach, where a tour is divided into 12 one-unit stretches and one very long stretch:

$$[1]\ [2]\ [3]\ [4] \cdots [12]\ [13, \cdots, 39]$$

$$[3]\ [4]\ [1]\ [2] \cdots [12]\ [13, \cdots, 39].$$

Here we are looking for a local improvement because most of the tour will be unchanged, while the tours within a short stretch are rotated.

Held and Karp have a number of rules for how many times to try the various methods of cutting up, and they seem to get fairly good results on problems up to about size 50, that is, either the correct answer (known to be optimal by some other means) or a good approximation.

The large-scale Dantzig, Fulkerson, and Johnson problem was solved five times, with five different starting solutions. Two of the answers came out optimal (699); the others were quite close (704, 704, 705). For a 20-city problem, devised by Croes, three starting solutions gave optimal answers. A highly structured problem (a knight's tour on an

8 × 6 chess board) was attempted. The optimal solution is known to be 48. The answers obtained were 56, 52, 54, and 56.

Roughly speaking, the sort of computational results that were obtained by Held and Karp were: in the range of 20 to 25 cities, they almost always got the optimal solution, with running times of 2 to 5 minutes; in the range of 25 to 50 cities, they seemed to get good results, nearly optimal. Of course, "nearly optimal" is a little hard to define, but it suggests what the results look like. They are optimal about one out of every five times. The running times in this range are approximately five to fifteen minutes. Now this sort of result takes the traveling salesman problem, computationally, further than it has been taken before. It does not tie in very much with the polyhedral approach or with methods for the assignment problem. The next method to be described is a little more in that direction.

This is the branch-and-bound method, due to Little *et al.* (1963). The relevant figure here is Figure 8, which shows a node marked "all tours."

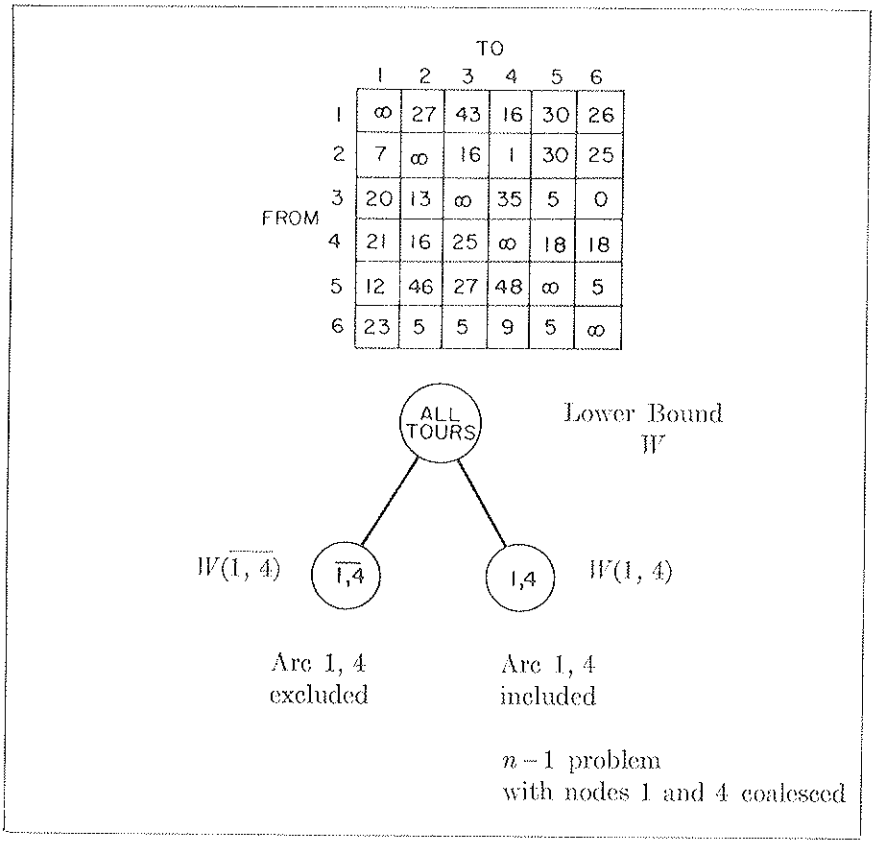


FIGURE 8. Branch and bound

By some scheme not yet specified, we find a lower bound W on the cost of all tours; that is, the optimal tour must be at least W. Then we select a particular arc—the arc 1, 4 is an example—and divide the tours up into two groups: those which use the arc 1, 4 and those which do not, that is, those from which the arc in question is excluded. Again, for each of these new groups of tours, each one of which is indicated by a node, a new lower bound is computed. Any tour in the group will have a cost equal to or exceeding the associated lower bound. The idea is to go on branching in this way. As a convention, we will say that when we branch to the right we go to a new node in which an arc is included, and when we branch to the left we go to a new node from which an arc is excluded.

Once we have branched n times to the right, there is only one tour remaining in the node at the end of that branch. If the cost of that tour is less than the bound appearing on all the other nodes of the tree, that tour must be optimal. So the idea is to develop the tree until it reaches this state. There are two kinds of choices that can be made in the development of the tree: (1) deciding which node, or group of tours, is to be broken up next and (2) once the node has been specified, deciding which arc is the one on which inclusion and exclusion are to be based. Before making these decisions, we will briefly discuss the bound that is used.

This bound comes from doing the first few steps of the Hungarian method for the assignment problem. It is well known that if a constant is subtracted from any row or any column of the cost matrix, the solution doesn't change, in the sense that the same permutation is still the solution, although the *cost* associated with the solution does change. In fact, the cost of the optimal solution is reduced by the amount subtracted from the row or column. Therefore, if one subtracts numbers from rows and columns in such a way that what remains is nonnegative so that a tour using the remaining costs will still have a nonnegative cost, the total amount taken off will be a lower bound on the cost of any tour. This is the idea that is used to provide the bounds here and which is illustrated in Figure 9. The cost matrix is obtained from the cost matrix of Figure 8, taking as much as possible away from each row and each column. One takes the smallest element in the row and subtracts that from all of the other elements. Everything remaining is nonnegative, and the total amount taken off in this fashion from the original cost matrix is 48. This then is the lower bound for the node representing all tours.

Let us now consider branching. We will explain later why the particular node 1, 4 was chosen for branching. On the assumption that we do branch on node 1, 4, the tours split up into two groups: those using the arc 1, 4 and those in which its use is not permitted. Now once it is decided that a particular arc must be used, you may coalesce the two cities involved; so you are now dealing with an $(n - 1)$-dimensional traveling salesman

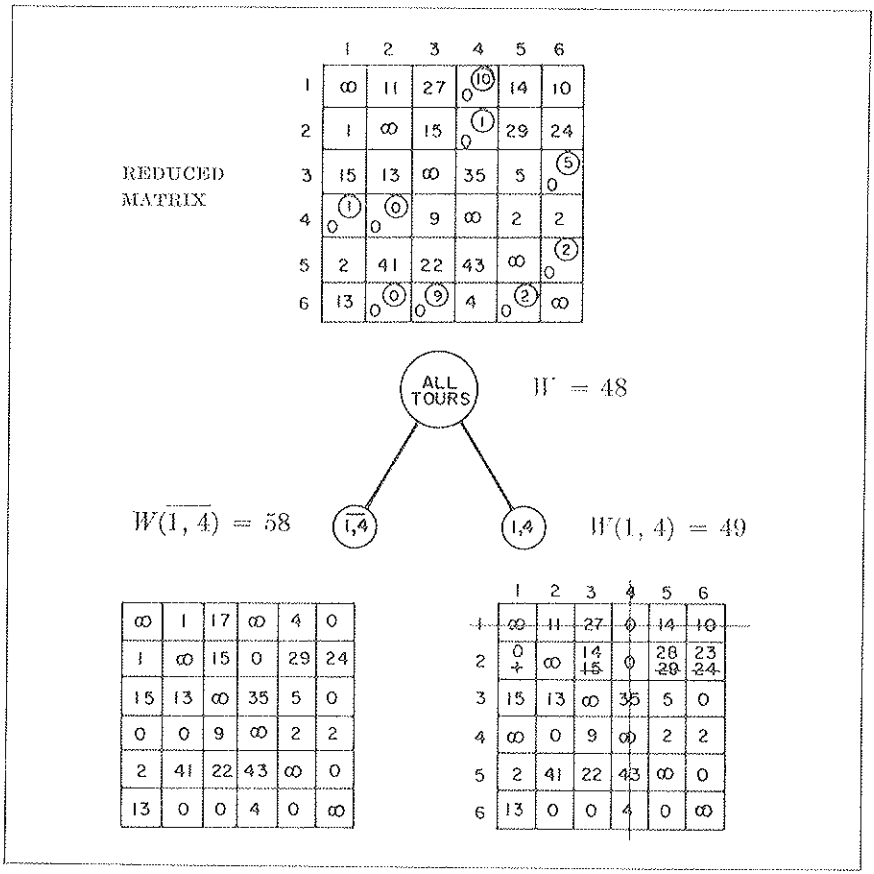


FIGURE 9. Two branching choices, branch to produce max change in left bound

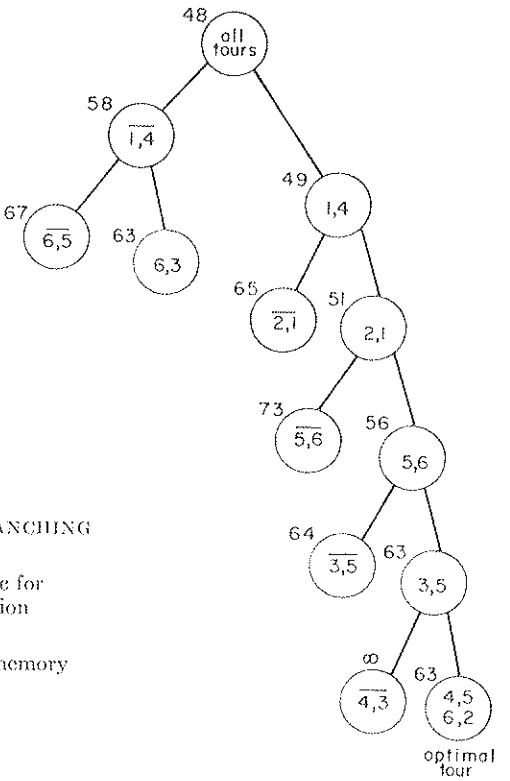
problem. This means that in the new cost matrix for the $(n-1)$-dimensional matrix, one can strike out, in our case, row 1 and column 4. Furthermore, since we are using the arc 1, 4, we will not use the arc 4, 1, which would complete a loop. So the arc 4, 1 can receive a cost of infinity. It may now be possible in this new, smaller matrix with changed costs to make further reductions and thus get a new lower bound. In this particular case (Figure 9), the new lower bound is 49. Similarly, on the other side in the other branch, the cost formerly attributed to the arc 1, 4 is changed to infinity, since that arc is now forbidden. This makes it possible to take 10 off the top row and still preserve the nonnegativity. Thus, the bound associated with this node goes from 48 to 58. One goes on in this fashion, updating the bound by using the new cost matrices obtained after each branching operation.

Let us now consider how to decide which arc to branch on. It is plausible to try to choose the branching arc so that the new bound associated with the lefthand node goes up as much as possible. The idea behind this is that we would like to find our optimal solution down one of the righthand paths, for that leads us into smaller and smaller problems and gets to a solution. We do not want to have to pursue lefthand branches, because the problem size there remains large. So the idea is to do the branching in such a way that the bound on the left goes up, and this branch is unlikely to obtain the optimal solution.

It is quite easy to check through a matrix and find out which branch gives the biggest increase in the lefthand bound (see Little *et al.*, 1963). It turns out that we want to branch only on those arcs that have zeros, and it is not hard to pick from among those the one that does the most for the bound. We have branched according to that rule in the procedures shown in Figures 8 and 9.

There is still the question of which branches of the tree should be developed and which branches should be left alone. A very plausible thing to do is to pursue that branch that has the lowest bound associated with it. That would be where the low-cost tours can be found. So one rule to follow would be to branch on the node currently having the lowest bound. We may, of course, run into the problem that the tree becomes too large and that there may be too many nodes to hold in memory. This difficulty can be avoided by a different approach. This is to proceed by always branching to the right until we have reached a point where no further right branching is possible. Then back up and take one left, and then back up again and stay, so to speak, as far right in the tree as possible. This scheme would not be expected to be as fast as the one in which the choice of node is made on the basis of the lower bound, but it has the advantage that we can keep the number of stored nodes down to a small multiple of n. So there are two things we can do with the node choice. We can use it to promote a rapid calculation or to conserve memory. This second use is also illustrated in Figure 10.

Figure 11 shows some of the computational results. They are very good. The problems on which the test runs were made were random problems with costs drawn from a uniform distribution of three-digit numbers. The first column shows the number of cities in the problem, the second column shows the number of problems run, the third column shows the time in minutes of IBM 7090 time, and the fourth column shows the standard deviation of the running times. These results are very good because we are getting answers up to 40 cities; this is an area where people have not previously obtained optimal solutions. Even an incomplete calculation gives both a tour and a lower bound. A good rule of thumb for the running times involved here is that adding 10 cities multiplies the running time



CHOICE OF BRANCHING

(1) Good heuristic for rapid calculation

(2) To conserve memory

FIGURE 10

Uniformly distributed three-digit random distances

No. of cities	No. of problems	Mean T' (IBM 7090 minutes)	Standard deviation
10	100	0.012	.007
20	100	0.084	.063
30	100	0.975	1.280
40	5	8.37	10.2

FIGURE 11. Computational results

by 10. So, as far as computing is concerned, we are still dealing with an exponentially increasing method.

In view of this, it is especially comforting to note that this computation has the feature that if we do not finish, we can still leave it. This is because we have bounds on each node. The smallest of these bounds is an underestimate of the cost of the optimal solution. Thus, the gap between the smallest bound and the best solution we have at that point is an overestimate of the error we would make in accepting our current best tour as the optimal one.

So far I have discussed the general traveling salesman problem. I am not aware of much work having been done on special cases of the problem. Nevertheless, in sequencing problems, it is not true that we have to deal with every possible cost matrix. For example, here is one very special case. Consider making sandpaper in a sandpaper-making machine (Maxwell, 1962). When the machine finishes making coarse sandpaper, there is a great deal of coarse sand in the machine. If fine sandpaper is made next, the coarse sand must be taken out. On the other hand, if coarser sandpaper is made next, it doesn't matter that much. The changeover cost in this problem is the cost of taking the sand out of the machine. Clearly, here the problem is easily solved. You should start with the fine sandpaper and work steadily up to the coarser types. This situation is very special and very simple, but it is a solvable special case. What else can we do with special traveling salesman problems?

We next come to some work of P. C. Gilmore and myself on a special traveling salesman problem (Gilmore and Gomory, 1964). I will start by giving a particular example. Suppose that there are a number of jobs to be done on a furnace. Each job is loaded into the furnace at a certain starting temperature, is cycled through various different temperatures, and is taken out at a certain ending temperature. For example, in the problem of Figure 12, job 1 starts at a temperature of 500°, and at a later time it comes out leaving the furnace at 611°. The next job must be started at a temperature of 750°. So the furnace has to be heated up from 611° to 750° before the next job can go in. Job 2 ends at 400°, and we must now heat the furnace to 550° for job 3. The problem here is to arrange the jobs in such an order that the cost of changing the furnace temperature is minimized. That cost may be a delay in time, it may be the total energy involved, or anything of that sort. The general problem of this type, which we call the problem of sequencing a one state-variable machine, is quite similar to this special case. There is one variable which characterizes the state of the machine, and moving this variable up and down is what incurs changeover cost. More precisely, we will assume that we have n jobs, $J_1, \cdots, J_n$. Each J_i is characterized by two numbers, A_i and B_i; A_i can be regarded as the starting state of the machine and

FURNACE	Job 1	Starts	$T = 500°$
		Ends	$T = 611°$
	Job 2	Starts	$T = 750°$
		Ends	$T = 400°$
	Job 3	Starts	$T = 550°$
		.	.
		.	.
		.	.

Cost of changing T between jobs is to be minimized

GENERAL PROBLEM

Jobs $J_1, \cdots J_n$

$$J_i = (A_i, B_i)$$

$$\left\{\begin{aligned} c_{i,j} &= \int_{B_i}^{A_j} f(x)\,dx && \text{if} \quad A_j \geq B_i \\ c_{i,j} &= \int_{A_j}^{B_i} g(x)\,dx && \text{if} \quad A_j < B_i \end{aligned}\right\}$$

$$f(x) + g(x) \geq 0$$

FIGURE 12. Sequencing a one state-variable machine

B_i as the state in which the machine is left. If $A_j \geq B_i$, the cost $c_{i,j}$ of following i with j is given by

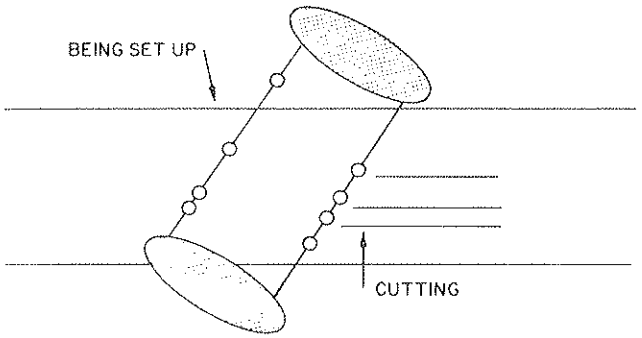
$$c_{i,j} = \int_{B_i}^{A_j} f(x)\,dx.$$

In other words, if we must go to a higher state, there is a cost $f(x)$ for every infinitesimal increase in the state variable. There is a similar cost if the state variable must be changed downward (that is, if $A_j < B_i$), and we can have a separate cost variable $g(x)$ for a downward change. Then,

$$c_{i,j} = \int_{A_j}^{B_i} g(x)\,dx.$$

The only important condition is that the sum $f(x) + g(x)$ be nonnegative.

One state-variable machines are not always as obvious as a furnace. Here is a different example of a one state-variable machine. Let us imagine a long ribbon of glass that is being cut up by a cutting device, Figure 13. The device consists of two axles and round knives on each axle. There



Job is a cutting pattern

$J_i = (A_i, B_i) =$ (setup time, run time)

if j follows i and $A_j \geq B_i$, delay $A_j - B_i$

if j follows i and $A_j < B_i$, no delay

FIGURE 13

is a succession of patterns for knife positions which must be used for cutting the correct amounts and the correct widths of glass. Each pattern must be set up on an axle; then, while this axle is actually running and cutting glass, another pattern can be set up on the other axle. The problem is to sequence the patterns so that there is always time to set up the next pattern while the previous one is running. All patterns have different running times and different setup times. The setup time is the A_i associated with the ith job, and the run time is the B_i. If we try to minimize the delay that is incurred when there is an inadequate setup time, then the cost equations given in the preceding paragraph assume the following forms:

$$f(x) = 1, \qquad g(x) = 0;$$

hence, when $A_j \geq B_i$ (delay),

$$c_{i,j} = \int_{B_i}^{A_j} 1 \, dx = A_j - B_i,$$

and when $A_j < B_i$ (no delay),

$$c_{i,j} = \int_{A_j}^{B_i} 0 \, dx = 0.$$

The one state-variable here is actually the amount of time that the cutting pattern currently being used for cutting has already been running. Thus, we can regard this cutting machine as a one state-variable machine just like the furnace, although in this case what the one state-variable should be is not quite as obvious.

Next we turn to a description of how the one state-variable sequencing problem can be solved. As you will see, this is a throwback, in a way, to the polyhedral systems which we discussed earlier.

The first step in this calculation is to solve the corresponding assignment problem. This will give us a point, in fact a vertex, on the polyhedron P. Next (Figure 14), we consider some interchanges $\alpha_{i,j}$ which will modify this permutation. Figure 14a shows the assignment problem. The interchange $\alpha_{i,j}$ is itself a permutation which sends node i into j, node j into i, and leaves all other nodes the same. The effect of applying α to a permutation ϕ is shown in Figure 14b. The effect of using $\alpha_{i,j}$ to modify ϕ is the same as adding a four-arc loop which is directed forward along the dashed lines and backward along the two connecting straight lines. Since ϕ and the modified permutation ϕ' differ only by this single cycle, they are neighboring points on the polyhedron P. The cost equation becomes

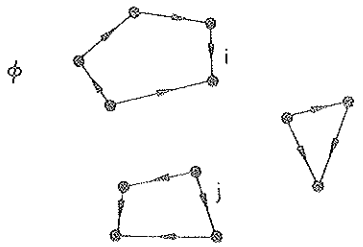
$$c(\alpha_{i,j}) = c(\phi\alpha_{i,j}) - c(\phi).$$

Thus, a series of such modifications will be a vertex-to-vertex exploration scheme. It is also clear that if such an interchange is effected on two nodes i and j in different subtours of the permutation ϕ, the result is to unite the two subtours into a single larger tour. Thus, we are moving closer to a single tour.

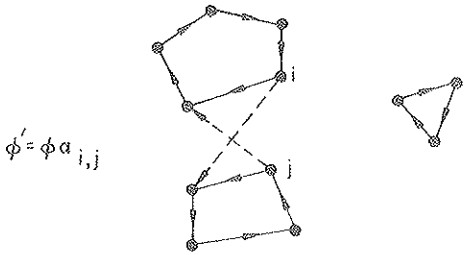
Next, we can associate costs with each interchange $\alpha_{i,j}$. The cost of $\alpha_{i,j}$ is the cost of $\phi\alpha_{i,j}$ less the cost of ϕ. Of course, this cost depends on ϕ as well as on $\alpha_{i,j}$. If $\alpha_{i,j}$ were applied to a different permutation, the cost would also be different.

[*Text resumes on page 114*]

(a) Solve Assignment Problem



(b) Consider Interchanges $\alpha_{i,j}$ and Their Costs



$$c(\alpha_{i,j}) = c(\phi\alpha_{i,j}) - c(\phi)$$

FIGURE 14. Method

We now consider a set of interchanges that will convert these permutations into a tour (Figure 15). If we simply drop the directions on the arcs in Figure 15a, we get Figure 15b, and if we put in a dark line whenever an interchange is applied, then the corresponding interchange connects the directed components in Figure 15a, just as the dark line connects the undirected components in Figure 15b. Consequently, a set of arcs that would connect up the components in Figure 15b corresponds to a set of interchanges which, if executed, would transform the permutation

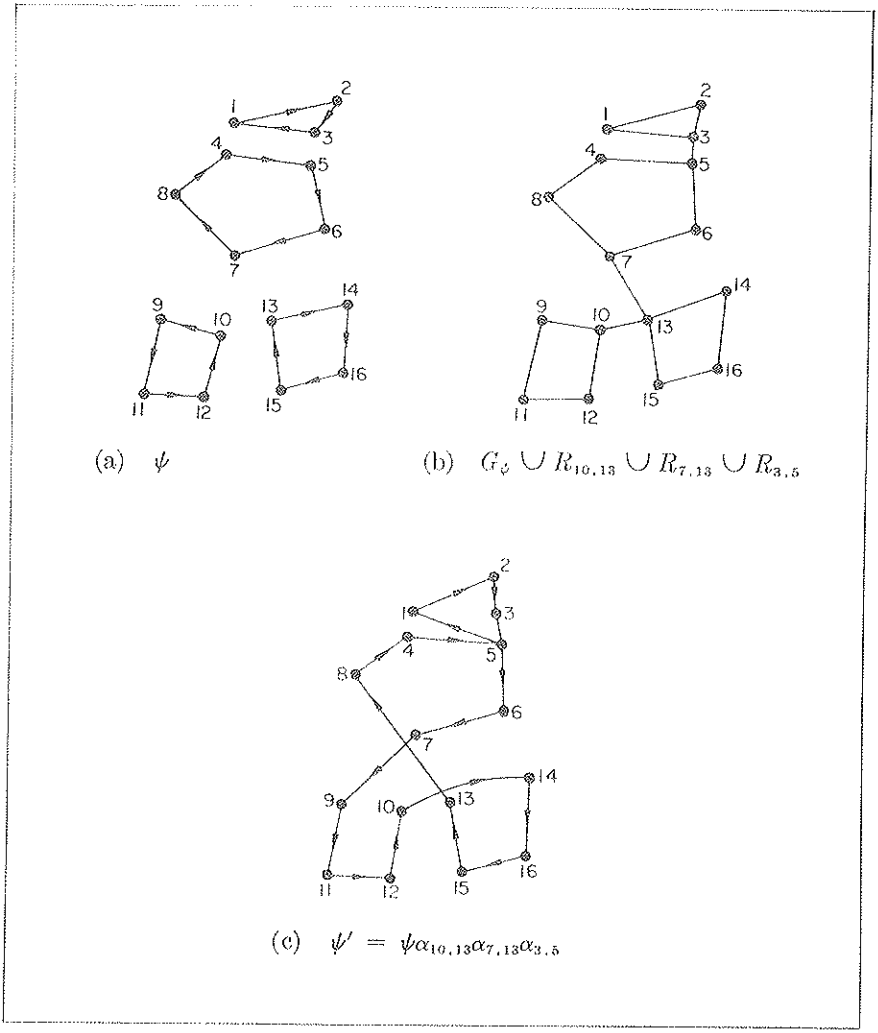


FIGURE 15

in Figure 15a into a tour such as that illustrated in Figure 15c. Now this suggests the following procedure: Put in the costs on all the arcs, and then choose the minimal-cost set of arcs that connect up the graph. Then perhaps the corresponding set of interchanges will, at least cost, transform the original permutation into a tour. This idea is illustrated in Figure 16, where the original arcs of the permutation appear as dashed lines. A number of interchanges are drawn as solid lines with their costs attached, and the bold solid lines are the minimal-cost interconnecting set. Fortunately, picking the minimal interconnecting set is a very slight modification on the well-known problem of picking a minimal-cost spanning tree. For this problem there is a well-known solution due to Kruskal (1957). So this is a solvable problem and one which can be done, in fact, extremely rapidly.

What we have said so far can be summarized as follows: It is plausible to pick a set of interchanges by a minimal spanning-tree argument and

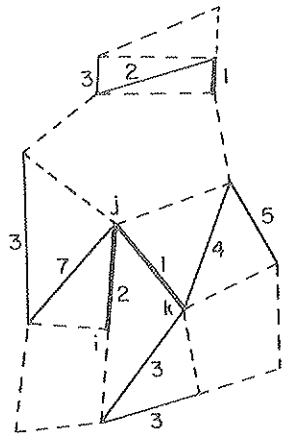


FIGURE 16

then to execute these interchanges to get what we hope will be a minimal-cost tour.

There are only two things wrong with this approach. First, the costs are generally wrong. When the interchange marked with a cost 7 is executed, it will not generally change the permutation cost by an amount 7. This is because, in general, the interchanges affect each other; the cost 7 was computed on the basis of the original permutation ϕ and will be different if the interchange is executed later, after several other interchanges have modified ϕ. Second, merely picking a set of interchanges, even if they are the right ones, does not specify a tour, because different tours will be obtained by executing the set of interchanges in a different order. Since, in general, these different resulting tours will have different costs, merely specifying a set of interchanges cannot possibly specify the optimal tour. It is necessary to have both a set of interchanges and a specified order in which to execute them before it is even possible to reach an answer.

These, then, are the difficulties which, in general, apply to this approach to the traveling salesman problem. However, in our particular case, we can get over these difficulties. In our case, the following three statements are correct, provided that the jobs are renumbered so that their final states B_i are numbered in increasing size:

First, the set of interchanges chosen by the minimal spanning-tree algorithm is correct and contains only interchanges of the form $\alpha_{i,i+1}$. Thus, it is meaningful to talk about the ith interchange, meaning $\alpha_{i,i+1}$.

Second, the minimal-cost tour will be obtained if this set of interchanges is executed in the correct order, that is, if we divide the interchanges into two groups. The ith interchange goes into group 1 if under the original assignment the ith job left the state variable so that it had to be increased for the start of the succeeding job. An interchange goes into group 2 if under the original assignment the corresponding job left a state variable that had to be decreased. Then a permutation ψ^* is obtained by applying to ϕ the interchanges of group 1 with the largest-indexed interchange first, the second-largest interchange second, etc., and then applying the interchanges of group 2, starting with the lowest-numbered interchange, following with the second interchange, etc. The resulting permutation ψ^* will be a cycle and will be the minimal-cost tour. The correct order is given by

$$\underbrace{\psi^*(i) = \phi \quad \alpha_{i_1,i_1+1}\alpha_{i_2,i_2+1}}_{\text{group 1: } A_{\phi(i)} \geq B_i} \quad \underbrace{\cdots \alpha_{j_1,j_1+1}\alpha_{j_2,j_2+1} \cdots \alpha_{j_m,j_m+1}\,(i)}_{\text{group 2: } A_{\phi(i)} < B_i},$$

where $i_1 > i_2$ and $j_1 < j_2$.

Third, the amount of computing that has to be done is very slight; it is $O(n^2)$.

By way of summary, we can see that the methods reviewed in this paper can be divided into three categories. First, there was a part where we dealt with linear programming and polyhedra. Then there was a part dealing with the work of Held and Karp and the branch-and-bound method. And finally there was a method that dealt with special $c_{i,j}$. Actually, I think that in all three areas which have been represented here, there is a great deal more work that can be done. In the area of linear programming and polyhedra, there are now methods available which should enable us to extend the more artistic, less systematic work already done. Certainly in the branching and dynamic programming area there are many more possibilities to be explored that could put us into a new range of practical applications, now that we know that present methods already take us into the range of 20 to 50 cities. In this area people will be producing one method after another, now that they know that a certain degree of success has already been obtained along these lines. Finally, it is hard to believe that there should be only one special case in the traveling salesman problem. This is an area which deserves to be looked at in more detail; a good look will probably reveal a variety of important and solvable special cases.

REFERENCES

Bellman, R. 1960. Combinatorial processes and dynamic programming *in* Combinatorial analysis (Proceedings of symposia in applied mathematics, vol. X), ed. R. Bellman and M. Hall, Jr. Providence, R. I.: Amer. Math. Soc., 217–49.

Gilmore, P. C., and R. E. Gomory. 1964. Sequencing a one state-variable machine: a solvable case of the traveling salesman problem. Operations Res., 12:655–79.

Held, M., and R. M. Karp. 1962. A dynamic programming approach to sequencing problems. J. Soc. Indust. Appl. Math., 10:196–210.

Heller, I. 1954. The travelling salesman's problem. Part I: Basic facts. Washington, D. C.: George Washington Univ., Logistics Research Project.

———. 1955. On the travelling salesman's problem *in* Proceedings of the second symposium in linear programming, vol. II. Washington, D. C.: Nat. Bur. Standards, 643–65.

Kruskal, J. B. 1957. On the shortest spanning subtree of a graph and the travelling salesman problem. J. Soc. Indust. Appl. Math., 5:32–38.

Kuhn, H. W. 1955. On certain convex polyhedra. Abstract 59-6-664. Bull. Amer. Math. Soc., 61.

Little, J. D. C., K. G. Murty, D. W. Sweeney, and C. Karel. 1963. An algorithm for the traveling salesman problem. Operations Res., 11:972–89.

Maxwell, W. L. 1962. The scheduling of economic lot sizes. Ithaca, N. Y.: Cornell Univ., Department of Industrial and Engineering Administration.

Motzkin, T. S. 1956. The assignment problem *in* Numerical analysis (Proceedings of symposia in applied mathematics, vol. VI), ed. J. H. Curtiss. New York: McGraw-Hill, 109–25.

[*Discussion begins on page 118*]

DISCUSSION

H. W. KUHN: I would like to make a correction to the number of faces cited in Dr. Gomory's paper for the nonsymmetric traveling salesman problem with five cities.

The correct enumeration, found by me in 1953, and presented at the summer meeting of the American Mathematical Society in Ann Arbor in 1955, includes 390 faces. That is to say, the miserable polyhedron has 24 vertices, is situated in eleven-dimensional space, but has 390 faces.

To turn this quibble in a more positive direction, let me describe an experiment that was run in 1953 with the help of Alan Hoffman. Imagine yourself sitting at the center of gravity of the polyhedron with a pistol. If you fire at random, the distance the bullet travels inside the polyhedron can be calculated by solving a linear program—namely, maximize the distance subject to the condition that the coordinates of the bullet be a convex combination of the vertices. In general, the vertices in the optimal basis will span a face, and thus, if you fire enough shots, you should find all types of faces.

In the actual experiment, the random directions were taken from a Los Angeles telephone book; the problems were sent to the National Bureau of Standards and solved by Saul Gass on the SEAC. Strangely enough, no matter how many times we tried it, we always hit the faces $x_{ij} = 0$. That is, out of 390 possible faces, our bullets always passed out through one class of 20 faces. This means to me that these are the "walls" of the polyhedron and that all the rest of the faces are small irregularities in the "corners" where these join. This is an empirical fact that has never received any theoretical explanation.

Of course, the same technique can be used to construct new constraints in any integer program in which the vertices are known explicitly.

S. H. CHASEN: I would like to compare the concept of randomness with its extreme opposite. I know that this will be repulsive after all of this fine analytical work, but let us introduce a little of the concepts of statistics. I think this is necessary for completeness. Suppose we take a number of nodes—for example, the 42 indicated cities of the United States. We pass through the cities randomly. With all of the different combinations, there are 42 factorial, which we might regard as infinite, for all practical purposes. It occurs to me that if we completely randomize our selection of tours from city to city, which is not elegant by any means, it does not take the computer very long to traverse around. We can do literally tens of thousands of tours with very nominal expenditure of computer time. Again, for the sake of completeness, how would this procedure, if we picked the minimum route or the minimum cost, compare with the optimal solution in terms of percentage of error?

R. E. GOMORY: I can only say that I do not know.

S. H. CHASEN: Could certain empirical investigations be done on a low scale?

R. E. GOMORY: Yes.

S. H. CHASEN: This is very important because, as the density of points increases, we might expect that there are a number of solutions which are very close to optimal—not optimal, of course, but, for all practical purposes, adequate.

R. E. GOMORY: This is an approach which people have tried on numerous scheduling problems. Sometimes it seems to be pretty good, and sometimes it seems to be pretty bad. I do not know of any experiments in this way on the traveling salesman problem. This does not mean that it would not work.

D. W. SWEENEY: There are some experiments going on now at the University of Rochester. Professor J. W. Gavett informed me that he is comparing the optimal solutions of job-shop setup problems with the rule-of-thumb solutions gained by taking as the next job the one with the shortest setup time. On a 25-city problem, less than 1,000 branches (partial or complete tours) were followed, using the branch-and-bound procedure to find an optimal tour. I would say that this is a far more hopeful approach than the use of random-walk techniques.

R. E. GOMORY: I would not be very optimistic about a scheme that went randomly from one city to another, because the salesman is then going to go to very distant points and ruin the tour.

S. H. CHASEN: A lot will depend on the confines of the area involved, will it not?

R. E. GOMORY: Yes, but there are bound to be some points that are distant.

A. J. GOLDSTEIN: I personally did the experiment that you have mentioned. I took an 11-city problem and exhibited all of the 1.8 million possible tours in numerical order. This took a minute and a half on the IBM 7094. The interval between the minimum and the median tours was divided into ten intervals, and I asked how many tours were in the first interval. I took several random 11-point examples from the unit square and found that this first interval had between 15 and 50 of the 1.8 million tours.

May I mention some results in the competition for speed and size of problem. Shen Lin at the Bell Telephone Laboratories has modified the dynamic programming approach so that on the IBM 7094 (which is 40% faster than the 7090) he has cut the time for a 13-city problem from 17 to 3.5 seconds. He has also used an iterative approach on the 20-, 25-, and 48-city problems discussed in the Held and Karp paper that you cited. The 25-city problem takes about 8 seconds to run, and the optimum

is found in 50% of the runs. For 48 cities the figures are 30 seconds and 30%. The running time is proportional to the cube of the number of cities.

J. Edmonds: I have a comment on the polyhedral approach to complete analysis, supplementing Professor Kuhn's remarks. I do not believe there is any reason for taking as a measure of the algorithmic difficulty of a class of combinatorial extremum problems the number of faces in the associated polyhedra. For example, consider the generalization of the assignment problem from bipartite graphs to arbitrary graphs. Unlike the case of bipartite graphs, the number of faces in the associated polyhedron increases exponentially with the size of the graph. On the other hand, there is an algorithm for this generalized assignment problem which has an upper bound on the work involved just as good as the upper bound for the bipartite assignment problem.

H. W. Kuhn: I could not agree with you more. That is shown by the unreasonable effectiveness of the Norman-Rabin scheme for solving this problem. Their result is unreasonable only in the sense that the number of faces of the polyhedron suggests that it ought to be a harder problem than it actually turned out to be. It is not impossible that some day we will have a practical combinatorial algorithm for this problem.

J. Edmonds: Actually, the amount of work in carrying out the Norman-Rabin scheme generally increases exponentially with the size of the graph.

The algorithm I had in mind is one I introduced in a paper submitted to the Canadian Journal of Mathematics (see Edmonds, 1965). This algorithm depends crucially on what amounts to knowing all the bounding inequalities of the associated convex polyhedron—and, as I said, there are many of them. The point is that the inequalities are known by an easily verifiable characterization rather than by an exhaustive listing—so their number is not important.

This sort of thing should be expected for a class of extremum problems with a combinatorially special structure. For the traveling salesman problem, the vertices of the associated polyhedron have a simple characterization despite their number—so might the bounding inequalities have a simple characterization despite their number. At least we should hope they have, because finding a really good traveling salesman algorithm is undoubtedly equivalent to finding such a characterization.

What amazes me is that the known successful examples of the philosophy I am propounding are so scarce. Perhaps a reason for the scarcity is the lack of attention which has been given to the theoretical distinction between finite algorithms and "better-than-finite" algorithms.

Practically the only other successful example that I know is the problem, mentioned by Dr. Gomory, of finding a minimum spanning tree in a graph which has a real numerical weight on each of its edges. The well-known, very easy algorithm for the problem is not usually regarded as an instance

of linear programming. However, the justification of the algorithm can be used to prove a simple characterization of the bounding planes of the polyhedron whose vertices correspond to the spanning trees of the graph. The number of these planes increases exponentially with the size of the graph. However, the algorithm can be interpreted as a refinement of a linear programming method operating on this polyhedron.

Dr. Gomory's general algorithm for integer programming may be regarded as a characterization of the bounding inequalities of the convex hull of the integer points in a prescribed convex polyhedron. The work involved in carrying out his algorithm depends not so much on the number of these inequalities as on the complexity of using the characterization to identify them.

R. E. Gomory: I think it is clear that the faces have nothing to do with it, particularly if we are not going to try something like a linear programming approach, which usually presupposes that we list all of the faces and equalities. In the spanning tree, we do not go at it in that way. It is not a measure of the inherent difficulty so much as it is a measure of what we have to face if we take the linear programming approach. Sometimes even the sheer number of faces is not a deterrent if we have some systematic way of getting at them. For example, if we took the cutting stock problem, which has millions of columns, and dualized that, we would have millions of faces; but if we generated the faces of the new problem by knapsack methods, we would get the same good computational results that we have on the primal problem. So the number of faces is not the problem. The question is whether we can get them. And the trouble with the traveling salesman problem is that we have not, up to now (I still think it can be done), been able to produce enough of them easily enough, whereas, if we have a thing with only a few faces, we can list them once and for all and then unleash the linear program.

DISCUSSION REFERENCE

Edmonds, J. 1965. Paths, trees, and flowers. Canad. J. Math., 17:449–67.