AN ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM

John D. C. Little

Massachusetts Institute of Technology

Katta G. Murty*

Indian Statistical Institute

Dura W. Sweeney†

International Business Machines Corporation

Caroline Karel

Case Institute of Technology

A 'branch and bound' algorithm is presented for solving the traveling
salesman problem. The set of all tours (feasible solutions) is broken up
into increasingly small subsets by a procedure called branching. For each
subset a lower bound on the length of the tours therein is calculated.
Eventually, a subset is found that contains a single tour whose length is
less than or equal to some lower bound for every tour. The motivation
of the branching and the calculation of the lower bounds are based on ideas
frequently used in solving assignment problems. Computationally, the
algorithm extends the size of problem that can reasonably be solved
without using methods special to the particular problem.

THE TRAVELING salesman problem is easy to state: A salesman
starting in one city, wishes to visit each of $n-1$ other cities once and
only once and return to the start. In what order should he visit the cities
to minimize the total distance traveled? For 'distance' we can substitute
time, cost, or other measure of effectiveness as desired. Distance or cost
between all city pairs are presumed known.

The problem has become famous because it combines ease of statement
with difficulty of solution. The difficulty is entirely computational, since
a solution obviously exists. There are $(n-1)!$ possible tours, one or more
of which must give minimum cost. (The minimum cost could conceivably
be infinite—it is conventional to assign an infinite cost to travel between
city pairs that have no direct connection.)

The traveling salesman problem recently achieved national prominence

when a soap company used it as the basis of a promotional contest. Prizes up to \$10,000 we[...] for [...]ing the most correct links in a particular 33-city problem. Quite a few people found the best tour. (The tie-breaking contest for these successful mathematicians was to complete a statement of 25 words or less on "I like ... because ...".) A number of people, perhaps a little over-educated, wrote the company that the problem was impossible—an interesting misinterpretation of the state of the art.

For the early history of the problem, see FLOOD.[1] In recent years a number of methods for solving the problem have been put forward. Some suffer from inefficiency, others produce solutions that are not necessarily optimal, and still others require intuitive judgments that would be hard to program on a computer. For a detailed discussion, see GONZALEZ.[2] We shall restrict our discussion to methods that (1) guarantee optimality, (2) seem reasonable to program, and (3) are general, i.e., not *ad hoc* to the specific numerical problem.

Among such methods the approach that has been carried furthest computationally is that of dynamic programming. HELD AND KARP[3] and and Gonzalez[2] have independently applied the method and have solved various test problems on computers. Gonzalez programmed an IBM 1620 to handle problems up to 10 cities. In his work the time to solve a problem grew somewhat faster than exponentially as the number of cities increased. A 5-city problem took 10 seconds, a 10-city problem took 8 minutes, and the addition of one more city multiplied the time by a factor, which, by 10 cities, had grown to 3. Storage requirements expanded with similar rapidity.

Held and Karp[3] have solved problems up to 13 cities by dynamic programming using an IBM 7090. A 13-city problem required 17 seconds. But such is the power of an exponential that, if their computation grows at the same rate as that of Gonzalez, a 20-city problem would require about 10 hours. Storage requirements, however, may become prohibitive before then. For larger problems than 13 cities, Held and Karp develop an approximation that seems to work well but does not guarantee an optimal tour.

We have found two papers in which the problem has been approached by methods similar to our 'branch and bound' algorithm. ROSSMAN, TWERY, AND STONE[4] in an unpublished paper apply ideas that they have called combinatorial programming.[5] To illustrate their method they present a 13-city problem. It was solved in 8 man-days. We have solved their problem by hand in about 3½ hours. EASTMAN,[6] in an unpublished doctoral thesis and laboratory report, presents a method of solution and several variations on it. His work and ours contain strong similarities. However, to use our terminology, his ways of choosing branches and of

calculating bounds are different from ours. He basically solves a sequence
of assignment problems that give his bounds. We have a simpler method,
and for branching we use a device with quite a different motivation. The
biggest problem Eastman solves is 10 cities and he gives no computation
times, so that effective comparisons are difficult to make.

Most published problems are symmetric, i.e., the distance from city i
to city j is the same as from j to i. The algorithm to be presented also
works for asymmetric problems; in fact, it seems to work better. Asymmetric problems arise in various applications. As an example from production scheduling, suppose that there is a production cycle of some time
period, during which an assembly line must produce each of n different
models. The cost of switching from model i to model j is c_{ij}. What order
of producing models minimizes total setup cost? This is a traveling salesman problem in which it would not necessarily be expected that $c_{ij} = c_{ji}$.

To summarize, 13 cities is the largest problem which we know about
that has been solved by a general method which guarantees optimality and
which can reasonably be programmed for a computer. Our method appreciably increases this number. However, the time required increases at
least exponentially with the number of cities and eventually, of course,
becomes prohibitive. Detailed results are given below.

THE ALGORITHM

THE BASIC method will be to break up the set of all tours into smaller and
smaller subsets and to calculate for each of them a lower bound on the cost
(length) of the best tour therein. The bounds guide the partitioning of the
subsets and eventually identify an optimal tour—when a subset is found
that contains a single tour whose cost is less than or equal to the lower
bounds for all other subsets, that tour is optimal.

The subsets of tours are conveniently represented as the nodes of a
tree and the process of partitioning as a branching of the tree. Hence we
have called the method 'branch and bound.'

The algorithm will simultaneously be explained and illustrated by a
numerical example. The explanation does not require reference to the
example, however, for those readers who wish to skip it.

Notation

The costs of the traveling salesman problem form a matrix. Let the
cities be indexed by $i = 1, \cdots, n$. The entry in row i and column j of the
matrix is the cost for going from city i to city j. Let

$$C = [c(i,j)] = \text{cost matrix}.$$

C will start out as the original cost matrix of the problem but will undergo
various transformations as the algorithm proceeds.

A tour, t, can be represented

$$t = [(i_1, i_2)(\cdots$$

which form a circuit going to
represents an arc or leg of the tr
is the sum of the matrix element
$z(t)$:

$$z(t) =$$

Notice that that t always picks
in each column. Also, let

$X, Y, \bar{Y} =$ nodes of the tree;
$w(X) =$ a lower bound on the
     for t a tour of X;
$z_0 =$ the cost of the best

Lower Bounds

A useful concept in construc
If a constant, h, is subtracted fr
the cost of any tour under the
This is because every tour must
that row. The relative costs of
any tour optimal under the old

The process of subtracting
element in the row will be call
negative elements and at least
called a *reduced matrix* and m
rows and columns. If $z(t)$ is
reduction, $z_1(t)$ the cost under
constants used in making the re

$$z$$

Since a reduced matrix contai
a lower bound on the cost of t

Consider then the 6-city pr
matrix by rows, then columns
duction is 48 so that $z(t) \geq 48$

Branching

The splitting of the set of
sented by the branching of a t
taining 'all tours' is self-expla
all tours which include the cit

from ours. He basically solves a sequence
.·e his bounds. We have a simpler method.
ice with quite a different motivation. The
·s is 10 cities and he gives no computation
·sons are difficult to make.
·re symmetric, i.e., the distance from city i
· to i. The algorithm to be presented also
·s; in fact, it seems to work better. Asym·
·us applications. As an example from pro·
·at there is a production cycle of some time
·bly line must produce each of n different
· from model i to model j is c_{ij}. What order
· total setup cost? This is a traveling sales·
·l not necessarily be expected that $c_{ij} = c_{ji}$.
·the largest problem which we know about
·ral method which guarantees optimality and
·rammed for a computer. Our method ap·
·r. However, the time required increases at
·number of cities and eventually, of course,
·l results are given below.

IE ALGORITHM

·break up the set of all tours into smaller and
·e for each of them a lower bound on the cost
·in. The bounds guide the partitioning of the
·ty an optimal tour—when a subset is found
·hose cost is less than or equal to the lower
·hat tour is optimal.
·conveniently represented as the nodes of a
·ioning as a branching of the tree. Hence we
··h and bound.'
·taneously be explained and illustrated by a
·planation does not require reference to the
·eaders who wish to skip it.

·; salesman problem form a matrix. Let the
·, n. The entry in row i and column j of the
·rom city i to city j. Let

$[c(i,j)] = $ cost matrix.

·l cost matrix of the problem but will undergo
·e algorithm proceeds.

A tour, t, can be represented as a set of n ordered city pairs, e.g.,

$$t = [(i_1, i_2)(i_2, i_3) \cdots (i_{n-1}, i_n)(i_n, i_1)],$$

which form a circuit going to each city once and only once. Each (i,j) represents an arc or leg of the trip. The cost of a tour, t, under a matrix, C, is the sum of the matrix elements picked out by t and will be denoted by $z(t)$:

$$z(t) = \sum_{(i,j) \text{ in } t} c(i,j).$$

Notice that that t always picks out one and only one cost in each row and in each column. Also, let

$X, Y, \bar{Y} = $ nodes of the tree;
$w(X) = $ a lower bound on the cost of the tours of X, i.e., $z(t) \geqq w(X)$
       for t a tour of X;
$z_0 = $ the cost of the best tour found so far in the algorithm.

Lower Bounds

A useful concept in constructing lower bounds will be that of reduction. If a constant, h, is subtracted from each element of a row of the cost matrix, the cost of any tour under the new matrix is h less than under the old. This is because every tour must contain one and only one element from that row. The relative costs of all tours are unchanged, however, and so any tour optimal under the old will be optimal under the new.

The process of subtracting the smallest element of a row from each element in the row will be called *reducing* the row. A matrix with non-negative elements and at least one zero in each row and column will be called a *reduced matrix* and may be obtained, for example, by reducing rows and columns. If $z(t)$ is the cost of a tour t under a matrix before reduction, $z_1(t)$ the cost under the matrix afterward, and h the sum of constants used in making the reduction, then

$$z(t) = h + z_1(t). \tag{1}$$

Since a reduced matrix contains only nonnegative elements, h constitutes a lower bound on the cost of t under the old matrix.

Consider then the 6-city problem shown in Fig. 1. Reduction of the matrix by rows, then columns, gives the matrix of Fig. 2. The total reduction is 48 so that $z(t) \geqq 48$ for all t.

Branching

The splitting of the set of all tours into disjoint subsets will be represented by the branching of a tree, as illustrated in Fig. 3. The node containing 'all tours' is self-explanatory. The node containing i,j represents all tours which include the city pair (i,j). The node containing $\overline{i,j}$ repre-

John D. C. Little, et al.

When a node X branches into
newly committed city pair will fre
the newly forbidden city pair $\bar{Y}$.

Flow Chart
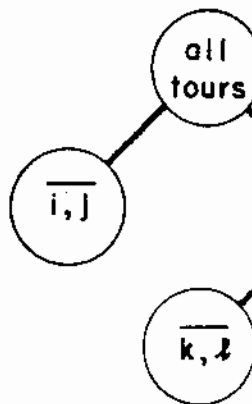
The workings of the algorithm v
flow chart of Fig. 4.

To

	1	2	3	4	5	6
1	∞	27	43	16	30	26
2	7	∞	16	1	30	25
3	20	13	∞	35	5	0
4	21	16	25	∞	18	18
5	12	46	27	48	∞	5
6	23	5	5	9	5	∞

(From, rows 1–6)

Fig. 1. Cost matrix for a 6-city problem. A typical tour might be
$t = [(1,3)(3,2)(2,5)(5,6)(6,4)(4,1)]$, which has the cost (length) $z = 43+13+30+5+9+21 = 121$.

sents all tours that do *not*. At the i,j node there is another branching. The
node containing $\overline{k,l}$ represents all tours that include (i,j) but not (k,l),
whereas k,l represents all tours that include both (i,j) and (k,l). In
general, by tracing from a node, X, back to the start, we can pick up which
city pairs are committed to appear in the tours of X and which are for-
bidden from appearing. If the branching process is carried far enough,
some node will eventually represent a single tour. Notice that at any stage

Fig. 3. S

	1	2	3	4	5	6
1	∞	11	27	0⑩	14	10
2	1	∞	15	0①	29	24
3	15	13	∞	35	5	0⑤
4	0①	0⑩	9	∞	2	2
5	2	41	22	43	∞	0②
6	13	0⑩	0⑨	4	0②	∞

Fig. 2. Cost matrix after reducing rows and columns. Circled numbers
are values of $\theta(i,j)$.

Box 1 starts the calculation by
problem into C, setting $X = 1$ to rep
the cost of the best tour so far to in

Box 2 reduces the matrix and lab

Box 3 selects (k,l), the city pair
The goal in doing this is to split th
quite likely to include the best tou
quite unlikely to include it. Possib
those involving an (i,j) for which $c($

Consider therefore the costs for
possible tours for $\bar{Y}$. Since city i i
tours must incur at least the cost of t
$c(i,j)$. Since city j must connect
least the cost of the smallest eleme
the sum of these two costs $\theta(i,j)$.
pair that gives the largest $\theta(i,j)$.
such that $c(i,j) = 0$, since otherwis

5	6
30	26
30	25
5	0
18	18
∞	5
5	∞

A typical tour might be
cost (length) $z = 43 + 13 +$

is another branching. The
nclude (i,j) but not (k,l),
both (i,j) and (k,l). In
start, we can pick up which
rs of X and which are for-
ocess is carried far enough,
ir. Notice that at any stage

5	6
14	10
29	24
5	⑤ 0
2	2
∞	② 0
⓪ ②	∞

d columns. Circled numbers
.j).

d the process, the union of the sets represented by the terminal nodes is the
set of all tours.

When a node X branches into two further nodes, the node with the
newly committed city pair will frequently be called Y and the node with
the newly forbidden city pair $\bar{Y}$.

Flow Chart

The workings of the algorithm will be explained by tracing through the
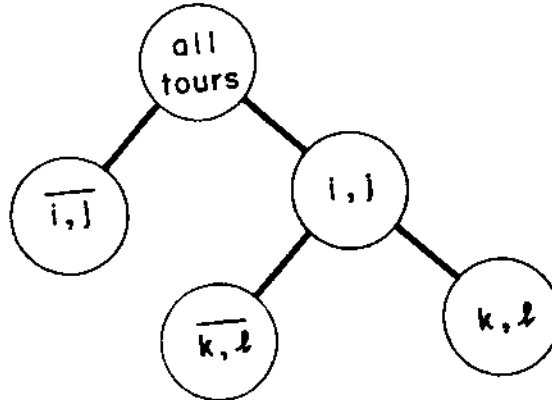flow chart of Fig. 4.

Fig. 3. Start of tree.

Box 1 starts the calculation by putting the original cost matrix of the
problem into C, setting $X = 1$ to represent the node, 'all tours,' and setting
the cost of the best tour so far to infinity.

Box 2 reduces the matrix and labels node X with its lower bound $w(X)$.

Box 3 selects (k,l), the city pair on which to base the next branching.
The goal in doing this is to split the tours of X into a subset (Y) that is
quite likely to include the best tour of the node and another $(\bar{Y})$ that is
quite unlikely to include it. Possible low cost tours to consider for Y are
those involving an (i,j) for which $c(i,j) = 0$.

Consider therefore the costs for tours that do not contain (i,j), i.e.,
possible tours for $\bar{Y}$. Since city i must be reached from *some* city, these
tours must incur at least the cost of the smallest element in row i, excluding
$c(i,j)$. Since city j must connect to *some* city, the tours must incur at
least the cost of the smallest element in column j, excluding $c(i,j)$. Call
the sum of these two costs $\theta(i,j)$. We shall choose (k,l) to be that city
pair that gives the largest $\theta(i,j)$. [This amounts to a search over (i,j)
such that $c(i,j) = 0$, since otherwise $\theta(i,j) = 0$.] Notice that, if $c(i,j)$ is
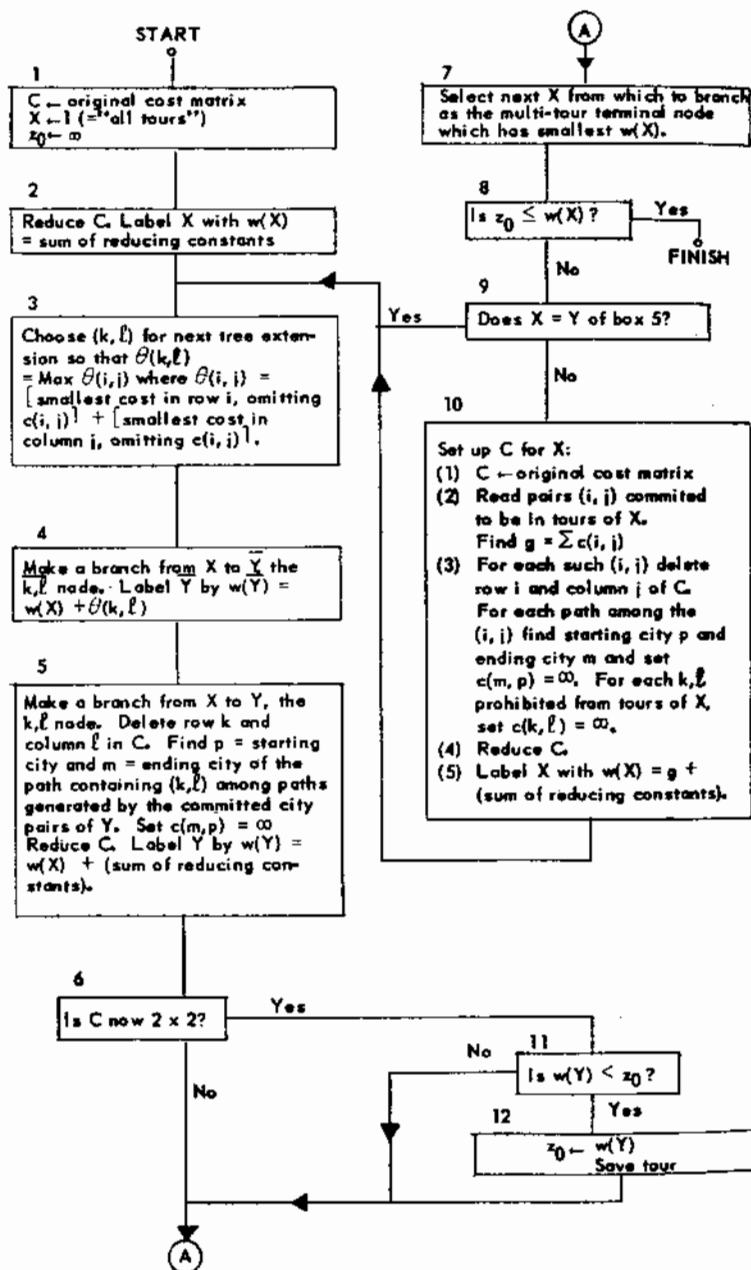
START

1
C ← original cost matrix
X ← l (="all tours")
$z_0 ← \infty$

2
Reduce C. Label X with w(X)
= sum of reducing constants

3
Choose (k, ℓ) for next tree exten-
sion so that $\theta(k,\ell)$
= Max $\theta(i, j)$ where $\theta(i, j)$ =
[smallest cost in row i, omitting
c(i, j)] + [smallest cost in
column j, omitting c(i, j)].

4
Make a branch from X to $\bar{Y}$ the
k,ℓ node. Label Y by w(Y) =
w(X) + $\theta(k, \ell)$

5
Make a branch from X to Y, the
k,ℓ node. Delete row k and
column ℓ in C. Find p = starting
city and m = ending city of the
path containing (k,ℓ) among paths
generated by the committed city
pairs of Y. Set $c(m,p) = \infty$
Reduce C. Label Y by w(Y) =
w(X) + (sum of reducing con-
stants).

6
Is C now 2 × 2?   Yes

No

(A)

7
Select next X from which to branch
as the multi-tour terminal node
which has smallest w(X).

8
Is $z_0 \le$ w(X)?   Yes

No   FINISH

9
Does X = Y of box 5?   Yes

No

10
Set up C for X:
(1) C ← original cost matrix
(2) Read pairs (i, j) commited
    to be in tours of X.
    Find g = Σ c(i, j)
(3) For each such (i, j) delete
    row i and column j of C.
    For each path among the
    (i, j) find starting city p and
    ending city m and set
    c(m, p) = ∞. For each k, ℓ
    prohibited from tours of X,
    set c(k, ℓ) = ∞.
(4) Reduce C.
(5) Label X with w(X) = g +
    (sum of reducing constants).

11
Is w(Y) < z_0?   No

Yes

12
$z_0 ← $ w(Y)
Save tour

(A)

Fig. 4. Flow chart of the algorithm.

set to infinity and then row i a[...]
ducing constants is $\theta(i,j)$[...]

For the example, the $\theta(i,j)$ [...]
the cells of the zeros of Fig. 2.[...]
(1,4) will be the first city pair [...]

Box 4 extends the tree from[...]
$w(\bar{Y}) = w(X) + \theta(k,l)$. In the[...]
is so labeled in Fig. 5b.

Box 5 sets up Y. Since th[...]
tours, row k and column l are[...]
Next, notice that (k,l) will be[...]
the city pairs that have been c[...]
path starts at city p and ends a[...]
The connecting of m to p shoul[...]
(a circuit with less than n cit[...]
Therefore, set $c(m,p) = \infty$.

After these modifications ([...]
places: row m, column p, any[...]
rows that had a zero in colum[...]
some zero that cannot have be[...]
reducing constants. The lowe[...]

$$w([...]$$

The algorithm operates so t[...]
in Box 3 with a matrix C and [...]
relation. If t is any tour of X[...]
the city pairs of t left after re[...]
and $z_1(t_1)$ the cost of t_1 under C[...]

$$z(t)[...]$$

This expression is true for the[...]
bound $w(X_1)$ and matrix C_1 of [...]
$w(X_2)$ and reduced matrix C_2 [...]
out of X_1.) It will be shown [...]
true for X_2.

The operations on C_1 to ge[...]
of the form: delete row i and co[...]
of X_2, insert various infinities,[...]
form

$$w(X_2) =[...]$$

where the summation is over t[...]
and h is the sum of the reducin[...]

..t to infinity and then row i and column j are reduced, the sum of the reducing constants is $\theta(i,j)$.

For the example, the $\theta(i,j)$ values are written in small circles placed in he cells of the zeros of Fig. 2. The largest θ is $\theta(1,4)=10+0=10$ and so 1,4) will be the first city pair used for branching.

Box 4 extends the tree from node X to $\bar{Y}$. As will be shown below, $r(\bar{Y})=w(X)+\theta(k,l)$. In the example $w(\bar{Y})=10+48=58$ and the node is so labeled in Fig. 5b.

Box 5 sets up Y. Since the city pair (k,l) is now committed to the tours, row k and column l are no longer needed and are deleted from C. Next, notice that (k,l) will be part of some connected path generated by he city pairs that have been committed to the tours of Y. Suppose the path starts at city p and ends at city m. (Possibly $p=k$ or $m=l$ or both.) The connecting of m to p should be forbidden for it would create a subtour a circuit with less than n cities) and no subtour can be part of a tour. Therefore, set $c(m,p)=\infty$.

After these modifications C can perhaps be reduced in the following places: row m, column p, any columns that had a zero in row k, and any rows that had a zero in column l. All other rows and columns contain some zero that cannot have been disturbed. Let h be the sum of the new reducing constants. The lower bound for Y will now be shown to be

$$w(Y)=w(X)+h.$$

The algorithm operates so that the investigation of each node, X, starts in Box 3 with a matrix C and a lower bound $w(X)$ that stand in a special relation. If t is any tour of X, $z(t)$ its cost under the original matirx, t_1, the city pairs of t left after removing those committed to the tours of X, and $z_1(t_1)$ the cost of t_1 under C, then it will be shown that

$$z(t)=w(X)+z_1(t_1). \qquad (2)$$

This expression is true for the first node by (1). Suppose that from a bound $w(X_1)$ and matrix C_1 of a node X_1, the algorithm constructs a bound $w(X_2)$ and reduced matrix C_2 for a node X_2. (X_2 will be on some branch out of X_1.) It will be shown that, if (2) is true for X_1, (2) will also be true for X_2.

The operations on C_1 to get C_2 (shown in Boxes 5 and 10) are always of the form: delete row i and column j for each (i,j) committed to the tours of X_2, insert various infinities, reduce. The lower bound is always of the form

$$w(X_2)=w(X_1)+\sum c_1(i,j)+h, \qquad (3)$$

where the summation is over the city pairs committed in X_2 but not in X_1 and h is the sum of the reducing constants. But consider any t in X_2 (and

therefore in X_1). If we let $z_1(t_1)$ be the cost of the uncommitted city pairs of X_1 under C_1 and $z_2(t_2)$ be the cost of the uncommitted city pairs of X_2 under C_2,
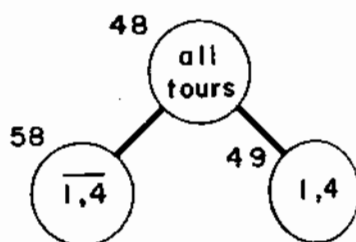
$$z_1(t_1) = \sum c_1(i,j) + h + z_2(t_2),$$

or using (2), assumed true for X_1,

$$z(t) = w(X_1) + \sum c_1(i,j) + h + z_2(t_2)$$
$$= w(X_2) + z_2(t_2),$$

so that (2) is true for X_2, as was to be shown.

Fig. 5. (a) Matrix after deletion of row 1 and column 4. (b) First branching.

Equation (3) is used to calculate the lower bounds in Boxes 4, 5, and 10. That these lower bounds are valid is established by (2) and the non-negativity of the elements of C.

For the example, the matrix of Fig. 5 shows the deletion of row 1 and column 4. The connected path containing (1,4) is (1,4) itself, so that $(m,p) = (4,1)$ and we set $c(4,1) = \infty$. Looking for reductions, we find that row 2 can be reduced by 1. Then $w(Y) = 48 + 1 = 49$ as shown.

It may be worth giving another example of finding (m,p). Suppose the committed city pairs were (2,1)(1,4)(4,3), and (5,6) and (k,l) were (1,4). Then the connected path containing (k,l) would start at 2 and end at 3 to yield $(m,p) = (3,2)$.

Box 6 checks to see whether a single tour node is near.

Box 7 selects the next *node* for branching. There are a number of ways the choice might be made. The way shown here is to pick the node with the smallest lower bound. This leads to the fewest nodes in the tree.
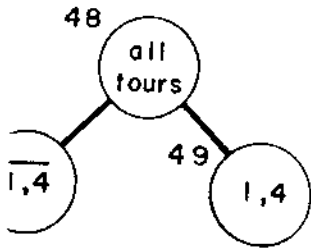
Box 8 checks to see whether the algorithm is finished—whether the
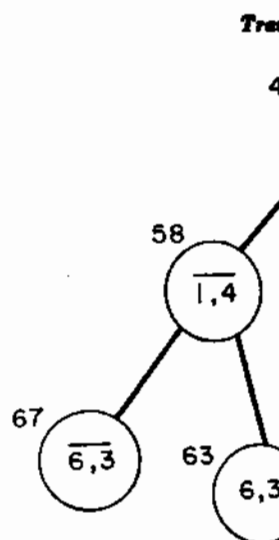
f the uncommitted city pair-
ncommitted city pairs of X;

$z_2(l_2),$

$h + z_2(l_2)$

(b)

mn 4. (b) First branching.

ounds in Boxes 4, 5, and 10.
hed by (2) and the non-

the deletion of row 1 and
,4) is (1,4) itself, so that
or reductions, we find that
= 49 as shown.

nding (m,p). Suppose the
(5,6) and (k,l) were (1,4).
start at 2 and end at 3 to

e is near.

here are a number of ways
e is to pick the node with
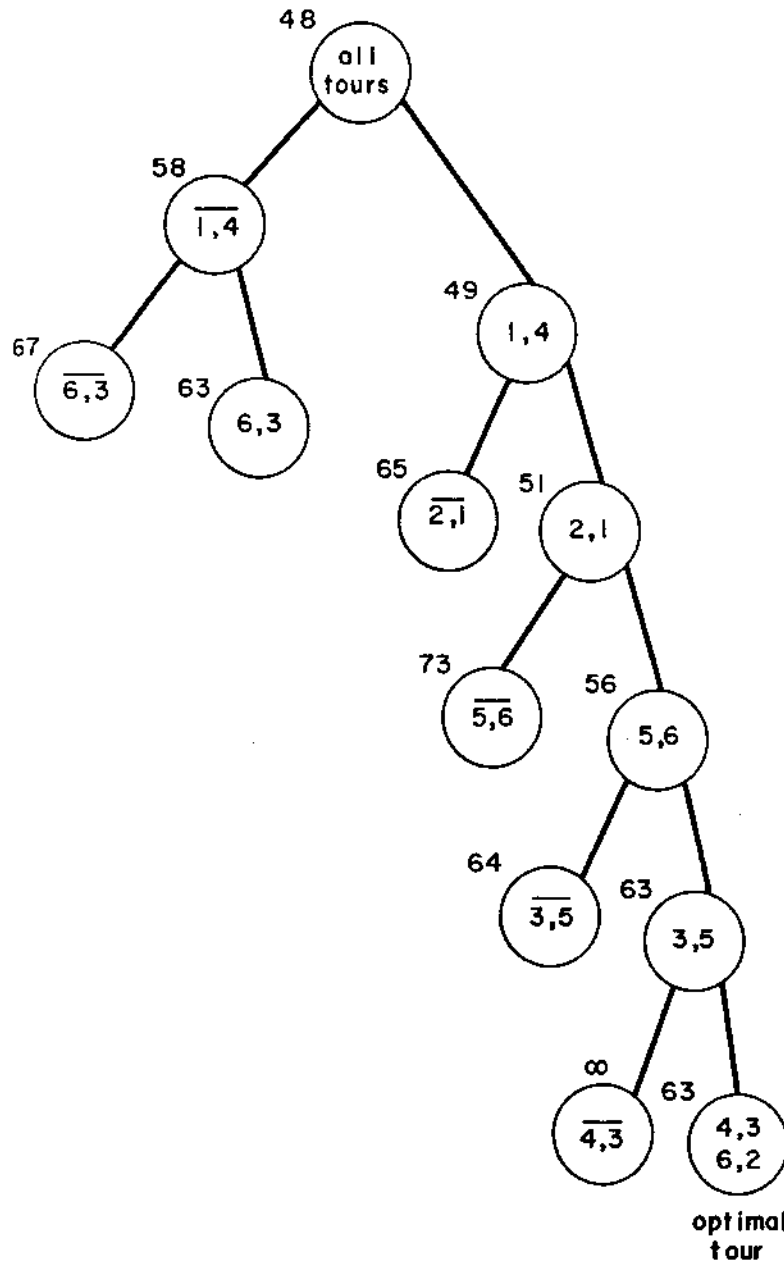est nodes in the tree.

is finished—whether the

Fig. 6. Final tree.

best tour so far has a cost less than or equal to the lower bounds on all terminal nodes of the tree.

Box 9 is a time saver. Most branching is from Y nodes, i.e., to the right. Such branching involves crossing out rows and columns and other manipulations that can be done on the matrix left over from the previous branching. When this case occurs, Box 9 detects it and the algorithm returns directly to Box 3.

Box 10 takes up the alternate case of setting up an appropriate lower bound and reduced matrix for an arbitrary X. Starting from the original cost matrix, rows and columns are deleted for city pairs committed to the tours of X, infinities are placed to block subtours and at forbidden city pairs, and the resulting matrix is reduced. The lower bound can be computed from (3) by thinking of X_1 in (3) as a starting node with $w(X_1)=0$ and matrix equal the original cost matrix. Since different ways of reducing a matrix may lead to different sums for the reducing constants, the recalculated $w(X)$ is substituted for the former one.

Boxes 11 and 12 finish up a single tour node. By the time C is a 2×2 matrix, there are only two feasible (i,j) left and they complete a tour. Since the box is entered with a reduced matrix, the costs of the final commitments are zero, and $z=w(Y)$ by (2). If $z<z_0$, the new tour is the best yet and is read off the tree to be saved.

Returning to the example, Box 7 picks 1,4 as the second node for branching and, since this is a branching to the right, C is already available in reduced form. As shown in Fig. 6, the next branching is on the basis of $(2,1)$ with $(m,p)=(4,2)$. Next, we go to the right from 2,1 on the basis of $(5,6)$ with $(m,p)=(6,5)$ and then from 5,6 on the basis of $(3,5)$ with $(m,p)=(6,3)$. At this point C is a 2×2 matrix, and we jump to Box 11 to finish the tour. We find $z=63$, which is stored as z_0 but, on returning to Boxes 7 and 8, we see that $\overline{1,4}$ has a lower bound of 58. To set up this node we go through Box 10. After the next branching, however, Box 8 shows that the problem is finished.

Discussion

At this point, let us stand back and review the general motivation of the algorithm. It proceeds by branching, crossing out a row and column, blocking a subtour, reducing the cost matrix to set a lower bound and then repeating. Although it is clear that the optimal solution will eventually be found, why should these particular steps be expected to be efficient? First of all, the reduction procedure is an efficient way of building up lower bounds and also of evoking likely city pairs to put into the tour. Branching is done so as to maximize the lower bound on the $\overline{k,l}$ node without worrying too much about the k,l node. The reasoning here is that the k,l node represents

lower bounds on all

Y nodes, i.e., to the
d columns and other
er from the previous
t and the algorithm

in appropriate lower
ing from the original
irs committed to the
nd at forbidden city
bound can be com
node with $w(X_1) = 0$
ent ways of reducing
onstants, the recalcu

the time C is a 2×2
ey complete a tour.
osts of the final com
new tour is the best

and node for branch
ady available in re
on the basis of $(2,1)$
on the basis of $(5,6)$
) with $(m,p) = (6,3)$.
11 to finish the tour.
; to Boxes 7 and 8,
node we go through
ws that the problem

al motivation of the
a row and column,
ver bound and then
n will eventually be
be efficient? First
ing up lower bounds
Branching is done
thout worrying too
k,l node represents

a smaller problem, one with the *k*th row and *l*th column crossed out. By putting the emphasis on a large lower bound for the larger problem, non-optimal tours are ruled out faster.

Insight into the operation of the algorithm is gained by observing that the crossing out of a row and column and the blocking of the corresponding subtour creates a new traveling salesman problem having one fewer city. Using the notation of Box 5, we can think of city m and city p as coalesced into a single city, say, m'. Setting $c(m,p) = \infty$ is the same as setting $c(m',m') = \infty$. The blocking of subtours is a way of introducing the tour restrictions into what is otherwise an assignment problem and is accomplished rather successfully by the algorithm.

Finally, unlike most mathematical programming algorithms, the one here has an extensive memory. It is not required that a trial solution at any stage be converted into a new and better trial solution at the next stage. A trial branch can be dropped for a moment while another branch is investigated. For this reason there is considerable room for experiment in how the next branch is chosen. On the other hand the same property leads to the ultimate demise of the computation—for n sufficiently large there are just too many branches to investigate and a small increase in n is likely to lead to a large number of new nodes that require investigation.

MODIFICATIONS

A VARIETY of embellishments on the basic method can be proposed. We record several that are incorporated in the computer program used in later calculations. For the program listing itself (*see* SWEENEY.[9])

Go to the Right

It is computationally advantageous to keep branching to the right until it becomes obviously unwise. Specifically, the program always branches from the k,l node unless its lower bound exceeds or equals the cost of a known tour. As a result a few extra nodes may be examined, but usually there will be substantial reduction in the number of time-consuming setups of Box 10.

One consequence of the modification is that the calculation goes directly to a tour at the beginning. Then, if the calculations are stopped before optimality is proven, a good tour is available. There is also available a lower bound on the optimal tour. The bound may be valuable in deciding whether the tour is sufficiently good for some practical purpose.

Throw Away the Tree

A large problem may involve thousands of nodes and exceed the capacity of high-speed storage. Storage can be saved, although usually at

the expense of time, by noting that, at any point in the computation, the cost of the best tour so far sets an upper bound on the cost of an optimal tour. Let the calculation proceed by branching to the right (storing each terminal node) until a single tour is found with some cost, say, z_0. Normally, one would next find the terminal node with the smallest lower bound and branch from there. Instead, work back through the terminal nodes, starting from the single tour, and discard nodes from storage until one is found with a lower bound less than z_0. Then, branch again to the right all the way to a single tour or until the lower bound on some right-hand node builds to z_0. (If the branch goes to the end, a better tour may be found and z_0 assigned a new, lower value.) Repeat the procedure: again work up the branch, discarding terminal nodes with bounds equal or greater than z_0 until the first one smaller is found; again branch to the right, etc.

The effect of the procedure is that very few nodes need be kept in storage —something on the order of a few n. These form an orderly sequence stretching from the current operating node directly back to the terminal node on the leftmost branch out of 'all tours.'

As an illustration, consider the problem and tree of Fig. 6. The computation would proceed by laying out in storage the nodes $4,1$; $\overline{2,1}$; $\overline{5,6}$; and $\overline{3,5}$. At the next step we find a tour with $z_0 = 63$ and the obviously useless node $\overline{4,3}$. The tour is stored separately from the tree. Working up the branch, first $\overline{3,5}$ is discarded, then $\overline{5,6}$ and $\overline{2,1}$, but $\overline{1,4}$ has a bound less than z_0. Therefore, branching begins again from there. A node $\overline{6,3}$ is stored and then we find the node to the right has a bound equal (s) z_0 and may be discarded. Working back up the tree again, $\overline{6,3}$ is discarded and, since that was the only remaining terminal node, we are finished.

The procedure saves storage but sometimes increases computation time. If the first run to the right turns up a rather poor tour, i.e., large z_0, the criterion for throwing away nodes is too stiff. The calculation is forced to branch out from many nodes whose lower bounds actually exceed the cost of the optimal tour. The original method would never do this for it would never explore such nodes until it had finished exploring every node with a smaller bound. In the process, the optimal tour would be uncovered and so the nodes with larger bounds would never be examined.

Taking Advantage of Symmetry

If the traveling salesman problem is symmetric and t is any tour, another tour with the same cost is obtained by traversing the circuit in the reverse direction. Probably the most promising way to handle this is to treat the city pair (i,j) as not being ordered. This leads naturally to a new and somewhat more powerful bounding procedure. Although the

---

basic ideas are not changed
So far, we have not done it.

There is another way to
easy to incorporate into our
by modifying the nodes alon
the nodes with no city pairs
such a node, X, branches int
(k,l) forbidden. The revers
not be in Y for the presence
tour. Such of the reverse to
them by setting $c(l,k) = \infty$
Thus, a reverse tour is prohi
the extent of having one com

A Computational Aid

In both hand and machin
first finding, for each row k a

$$\alpha(k) = \text{the sec}$$

$$\beta(l) = \text{the sec}$$

Then $\theta(k,l) = \alpha(k) + \beta(l)$ for
computation the $\alpha(k)$ can be
matrix and the $\beta(l)$ as an ext
few problems, one can see that
need to search the whole matr
and columns need be examine

Other Possibilities

If desired, the algorithm c
solutions. Instead of discardi
until eventually all the termi
single tours with $z = z_0$. Our
modification because in some
great deal—suppose the cost n

Quite possibly, the average
the assignment problem for the
by the cost of the optimal assig
the assignment problem leave
the larger lower bound with
starting lower bound to the co
ng that may be expected. O
been extensive and has yielded
speeded up, but some others w

t in the computation, the
on the cost of an optimal
to the right (storing each
some cost, say, z_0. Nor-
with the smallest lower
·ack through the terminal
nodes from storage until
hen, branch again to the
bound on some right-hand
nd; a better tour may be
peat the procedure: again
s with bounds equal or
nd; again branch to the

les need be kept in storage
'orm an orderly sequence
:tly back to the terminal

tree of Fig. 6. The com-
he nodes 4,1; $\overline{2,1}$; $\overline{5,6}$; and
and the obviously useless
ie tree. Working up the
but $\overline{1,4}$ has a bound less
·m there. A node $\overline{6,3}$ is
bound equal (s) z_0 and may
$\overline{6,3}$ is discarded and, since
re finished.

creases computation time.
·or tour, i.e., large z_0, the
he calculation is forced to
s actually exceed the cost
never do this for it would
ploring every node with a
· would be uncovered and
examined.

ietric and t is any tour,
·aversing the circuit in the
: way to handle this is to
This leads naturally to a
·rocedure. Although the

asic ideas are not changed much, considerable reprogramming is required.
So far, we have not done it.

There is another way to take advantage of symmetry and this one is
easy to incorporate into our program. All reverse tours can be prohibited
by modifying the nodes along the leftmost branch of the tree. These are
the nodes with no city pairs committed but some forbidden. Suppose that
such a node, X, branches into nodes, Y, with (k,l) committed, and, $\bar{Y}$, with
(k,l) forbidden. The reverse tours of Y all have (l,k) in them. They can-
not be in Y for the presence of both (k,l) and (l,k) is not possible in any
tour. Such of the reverse tours as were in X are in $\bar{Y}$. We may prohibit
them by setting $c(l,k) = \infty$ [as well as $c(k,l) = \infty$] in any matrix for Y.
Thus, a reverse tour is prohibited as soon as the tour itself is identified to
the extent of having one committed city pair.

A Computational Aid

In both hand and machine computation $\theta(k,l)$ is easiest calculated by
first finding, for each row k and column l of the reduced matrix:

$$\alpha(k) = \text{the second smallest cost in row } k.$$

$$\beta(l) = \text{the second smallest cost in column } l.$$

Then $\theta(k,l) = \alpha(k) + \beta(l)$ for any (k,l) which has $c(k,l) = 0$. In a hand
computation the $\alpha(k)$ can be written as an extra column to the right of the
matrix and the $\beta(l)$ as an extra row at the bottom. After working out a
few problems, one can see that when the branching is to the right there is no
need to search the whole matrix to reset α and β, but that only certain rows
and columns need be examined.

Other Possibilities

If desired, the algorithm can be modified so as to generate all optimal
solutions. Instead of discarding nodes with $w(X) = z_0$, split them up further
until eventually all the terminal nodes either have $w > z_0$ or are optimal
single tours with $z = z_0$. Our computer program does not include this
modification because in some cases it will increase the computing time a
great deal—suppose the cost matrix were all zeroes.

Quite possibly, the average computing time can be decreased by solving
the assignment problem for the original cost matrix and reducing the matrix
by the cost of the optimal assignment in Box 2. (Some methods for solving
the assignment problem leave it in reduced form.) The advantage lies in
the larger lower bound with which the problem starts. The closer the
starting lower bound to the cost of the optimal tour, the less is the branch-
ing that may be expected. Our exploration of the possible gains has not
been extensive and has yielded mixed results: CROES' 20-city problem[8] was
speeded up, but some others were lengthened.

The idea that we are calling 'branch and bound' is more general than the traveling salesman algorithm. A minimal solution for a problem can be found by taking the set of all feasible solutions, splitting it up into disjoint subsets, finding lower bounds on the objective function for each subset, splitting again the subset with the smallest lower bound, and so forth, until an optimal solution is found. The efficiency of the process, however, rests very strongly on the devices used to split the subsets and to find the lower bounds. As a simple example of another use of the method, if the step of setting $c(m,p) = \infty$ is omitted from the traveling salesman algorithm, it solves the assignment problem. For another example, *see* DOIG AND LAND.[10]

TABLE I
MEAN AND STANDARD DEVIATION OF T FOR RANDOM DISTANCE MATRICES
(T = time in minutes used to solve Traveling Salesman Problem on IBM 7090.)

Number of cities	Number of problems solved	Mean T	Std. dev. T	Mean[a] log T	Std. dev.[a] log T
10	100	0.012	0.007	log 0.015	log 1.24
20	100	0.084	0.063	log 0.067	log 2.09
30	100	0.975	1.240	log 0.63	log 2.94
40	5	8.37	10.2	log 4.55	log 3.74

[a] Obtained by plotting the cumulative frequency on log normal probability paper and fitting a straight line, except in the 40-cities case for which the computation was numerical. In the case of 10 cities the log normal fits only the tail of the distribution—30 per cent of the problems went directly to the solution without extra branching and thereby produced a lump of probability at 0.002 minute.

CALCULATIONS

PROBLEMS UP to 10 cities can be solved easily by hand. Although we have made no special study of the time required for hand computations, our experience is that a 10-city problem can be solved in less than an hour.

The principal testing of the algorithm has been by machine on an IBM 7090. Two types of problems have been studied: (1) asymmetric distance matrices with elements consisting of uniformly distributed 3-digit random numbers and (2) various published problems and subproblems constructed therefrom by deleting cities. Most of the published problems have been made up from road atlases or maps and are symmetric.

The random distance matrices have the advantage of being statistically well defined. Average computing times are displayed in Table I and curve (a) of Fig. 7. Problems up to 20 cities usually require only a few seconds. The time grows exponentially, however, and by 40 cities is beginning to be appreciable, averaging a little over 8 minutes. As a rule of thumb, adding 10 cities to the problem multiplies the time by a factor of 10.

The standard de...
problem size as may...
is skew, the simple st...
purpose of estimati...

Fig. 7. Computin...
random number dist...
and Karp's 25-city ...
Karp's 48-city proble...

The standard deviation of these computing times also increases with problem size as may be seen in Table I. Because the distribution of times is skew, the simple standard deviation is a little misleading, at least for the purpose of estimating the probability of a long calculation. Conse-

:d bound' is more general than mal solution for a problem can lutions, splitting it up into dis objective function for each sub llest lower bound, and so forth, ficiency of the process, however, split the subsets and to find the other use of the method, if the he traveling salesman algorithm, nother example, *see* DOIG AND

RANDOM DISTANCE MATRICES
alesman Problem on IBM 7090.)

*.	Mean[a] $\log T$	Std. dev.[a] $\log T$
.	log 0.015	log 1.24
;	log 0.067	log 2.09
>	log 0.63	log 2.94
	log 4.55	log 3.74

cy on log normal probability paper and r which the computation was numerical. c tail of the distribution—30 per cent of t extra branching and thereby produced

ONS

sily by hand. Although we have d for hand computations, our ex solved in less than an hour.

has been by machine on an IBM studied: (1) asymmetric distance ormly distributed 3-digit random ems and subproblems constructed he published problems have been re symmetric.

ie advantage of being statistically ire displayed in Table I and curve isually require only a few seconds. and by 40 cities is beginning to be utes. As a rule of thumb, adding me by a factor of 10.

Fig. 7. Computing times on IBM 7090. (a) Average times for 3 digit random number distance matrices. (b) Subproblems derived from Held and Karp's 25-city problem. (c) Subproblems derived from Held and Karp's 48-city problem.

quently, a log normal distribution has been fitted to the tail of the distribution. A use of the tabulated numbers would be, for example, as follows: A two-sigma deviation on the high side in a 40-city problem would be a calculation that took $(3.74)^2(4.55) = 64$ minutes. In other words, the probability that a 40-city random distance problem will require 64 minutes or more is estimated to be 0.023.

Symmetric problems have usually taken considerably longer than random distance problems of the same size. To obtain a variety of problems of increasing size, we have taken published problems and abstracted subproblems of increasing size. The first 10 cities were taken, then the first 11 cities, etc., until the computing times became excessive. Curves (b) and (c) of Fig. 7 show the results for subproblems pulled out of the 25- and 48-city problems of Held and Karp.[3] The 25-city problem itself took 4.7 minutes. We note that Held and Karp's conjectured optimal solution is correct.

A few miscellaneous problems have also been solved. Croes'[8] 20-city problem took 0.126 minutes. A 64 'city' knight's tour took 0.178 minutes.

ACKNOWLEDGMENT

REFERENCES

1. M. M. FLOOD, "The Traveling Salesman Problem," *Opns. Res.* 4, 61–75 (1956).
2. R. H. GONZALEZ, "Solution of the Traveling Salesman Problem by Dynamic Programming on the Hypercube," Interim Technical Report No. 18, OR Center, M. I. T., 1962.
3. M. HELD AND R. M. KARP, "A Dynamic Programming Approach to Sequencing Problems," *J. Soc. Indust. and Appl. Math.* 10, 196–210 (1962).
4. M. J. ROSSMAN, R. J. TWERY, AND F. D. STONE, "A Solution to the Traveling Salesman Problem by Combinatorial Programming," mimeographed.
5. M. J. ROSSMAN AND R. J. TWERY, "Combinatorial Programming," presented at 6th Annual ORSA meeting, May 1958, mimeographed.
6. W. L. EASTMAN, "Linear Programming with Pattern Constraints," Ph.D. dissertation, Harvard University, July 1958; also, in augmented form: Report No. BL-20 The Computation Laboratory, Harvard University, July 1958.

7. G. B. DANTZIG, D
   Scale Traveling
8. G. A. CROES, "A
   Res. 6, 791–814
9. D. W. SWEENEY, "
   ing Salesman Pr
10. A. DOIG AND A. B
    gramming Probl

al.

itted to the tail of the dis
would be, for example, as
side in a 40-city problem
i5) = 64 minutes. In other
listance problem will require

i considerably longer than
To obtain a variety of prob-
ied problems and abstracted
cities were taken, then the
; became excessive. Curves
bproblems pulled out of the
The 25-city problem itself
Karp's conjectured optimal

en solved. Croes'[8] 20-city
ht's tour took 0.178 minutes.

NT

em," *Opns. Res.* 4, 61–75 (1956).
Salesman Problem by Dynamic
i Technical Report No. 18, OR

ramming Approach to Sequencing
th. 10, 196–210 (1962).
NE, "A Solution to the Traveling
amming," mimeographed.
atorial Programming," presented
i, mimeographed.
Pattern Constraints," Ph.D. dis-
also, in augmented form: Report
Harvard University, July 1958.

7. G. B. DANTZIG, D. R. FULKERSON, AND S. M. JOHNSON, "Solution of a Large
Scale Traveling Salesman Problem," *Opns. Res.* 2, 393–410 (1954).
8. G. A. CROES, "A Method for Solving Traveling Salesman Problems," *Opns.
Res.* 6, 791–814 (1958).
9. D. W. SWEENEY, "The Exploration of a New Algorithm for Solving the Travel-
ing Salesman Problem," M.S. Thesis, M. I. T., 1963.
10. A. DOIG AND A. H. LAND, "An Automatic Method of Solving Discrete Pro-
gramming Problems," *Econometrica* 28, 497–520 (1960).