KM 10 (1983)
pag 139 - 158

INTERFACULTEIT DER ACTUARIELE WETENSCHAPPEN EN ECONOMETRIE

SOLVING THE TRAVELING SALESMAN PROBLEM IN BASIC

PIETER FRIS

TON VOLGENANT

Abstract

An algorithm for the problem will be explained. It is based on minimal spanning trees as lower bounds together with weights on the distances to improve these bounds.

A Basic program has been developed for an Apple II computer, that solves Euclidean problems of up to 42 cities in a reasonable time within 16K memory.

Universiteit van Amsterdam

JODENBREESTRAAT 23
1011NH AMSTERDAM
THE NETHERLANDS

Introduction

One of the most fascinating problems for which computer programs are being developed is the traveling salesman problem (TSP for short). It can be described as: A traveling salesman, starting at his residence, wants to visit a number of cities, each exactly once, and return home afterwards. He wants to make his tour as short as possible.

The TSP is so fascinating as the number of possible tours, from which the shortest one has to be chosen, increases exponentially with the number of cities; more precisely: the number of possible tours through n cities is (n-1)!. A computer able to evaluate a million tours per second, needs 0.12 milliseconds to solve a 6-city problem. For a 15-city problem, it takes already over 24 hours, and a 33-city problem is only solved in 8.3×10^{19} centuries!

So for this enumeration approach we run into trouble: very quickly the problem gets too large to be solved by a computer, no matter how powerful. The successful methods available nowadays skip as many tours as possible before or during the search process. They are based on "implicit enumeration".

A Basic computer program for use on a microcomputer was shown in the article "The Infamous Traveling Salesman Problem", by R. Parry and H. Pfeffer (Byte, July 1981). Their method is simple, but not very effective: They introduced a 12-city problem that could be solved only just within an hour on their microcomputer (Sw TPC-6800 system).

We will give a similar Basic program that uses many properties of the TSP. Knowledge of the article mentioned above is not needed to understand this one. Compared to the mentioned results, computation times will be reduced considerably, and it will be possible to solve larger problems. For instance, the same 12-city problem was solved 37 times faster, and this factor increases sharply with the number of cities. Our program is also capable of solving TSP's with different beginning and ending location.

Applications

Many practical problems are in fact a TSP, e.g. how to schedule an optimal tour for a postman to empty pillar boxes. Another example is the following problem:

A mechanical soldering-bolt, moving above a print plate, has to solder a given number of points on that plate. We now ask for the shortest tour of the soldering-bolt above the print plate.

A less trivial application arises when a machine has to accomplish a number of tasks successively (e.g. producing certain colours of paint). Costs arise by switching over to another task (cleaning and filling paint reservoirs), depending on the two considered tasks (from red to green may give different costs than from red to yellow). We now ask for a production sequence (red-orange-yellow etc.) with minimal total switching costs.
In this example the tasks correspond to the cities, and the switching costs to the distances (not symmetric in general).

Method

Search procedure

To evaluate all tours systematically we use the same method as Parry and Pfeffer: from a given starting city we build up a chain of connected edges, with different cities on that chain (An edge is the direct connection between two cities). We call this a required chain. Cities not on the required chain are called free cities. When no free city is left, we connect the last and the first city on the required chain: we obtain a tour. From all tours considered we store the shortest one.

We terminate the construction of a required chain when we are sure that this chain will never generate a shorter tour than found thus far. Then we will proceed with a new required chain according to a "Last In, First Out" rule (see example below).

The advantage is obvious if the construction of a required chain has not to be finished completely. When six free cities are left at the moment of termination, we skip 6! = 720 tours at once.

Lower bounds

To illustrate the elimination of tours we give an example:
Assume a 7-city problem with city 1 the starting and ending point. Suppose 1-5-2-6 is the required chain with length 75 and 3, 4 and 7 are free cities. Suppose further that we know a tour with lenth 100, and that the shortest path from 6, over 3, 4 and 7, to city 1, (in any order) has at least length 30 (30 is a lower bound for the length of that path). In that case, any tour with 1-5-2-6 must at least have length 105, so 1-5-2-6 can never be part of an optimal tour. So we skip the six tours 1-5-2-6-(3-4-7)-1
1-5-2-6-(3-7-4)-1.
.
.
.

Adding the cities in numerical order, our next required chain will be 1-5-2-7. And after 1-5-2-7 has been evaluated completely, we will continue with 1-5-3, etc.

We shall look for a method to calculate lower bounds for those shortest paths mentioned above. The computation of that path is nothing but solving a TSP over the free cities, with fixed beginning and ending point, so it is sufficient to consider lower bounds for the TSP with no required chain involved. Intuitively it is clear that the higher our lower bounds (the closer we approximate the optimal TSP value), the more tours can be skipped in the search procedure.

We must realize that computation of lower bounds is only useful when they can be calculated much easier than the optimal value of the TSP itself. That this is possible will be clear in the following examples:

Suppose we have n cities, with distances d_{ij} $(i,j = 1,\ldots,n)$, that are symmetric $(d_{ij} = d_{ji})$. We take $d_{ii} = \infty$ $(i = 1,\ldots,n)$, for edge i-i is not allowed.

A possible lower bound for the optimal value of the TSP is n times the minimal number in the distance matrix, because every edge in a tour is at least as long as the minimum over all edges. As n^2 distances must be called from memory, this lower bound is calculated in order n^2 (i.e. the execution time for this bound increases quadratically with the number of cities). This is very fast compared to solving the TSP, but unfortunately this bound is bad.

A better bound (not more complex than the first one) is the sum over all row-minima in the distance matrix. This bound takes into account that every city must be left exactly once in a tour, so the distances to the next cities are at least as long as the shortest distance in each row.

1-trees

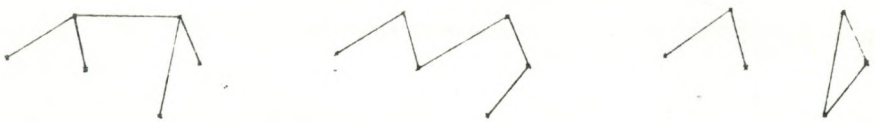
In our program we use 1-tree lower bounds, developed by Held and Karp in 1971. They can also be calculated in order n^2, and provide good results in practice. To describe what a 1-tree is, we give first two definitions:

1) A cycle is a chain of edges with the same beginning and ending point (figure 1).

A cycle containing all cities is a tour.

figure 1 : a cycle

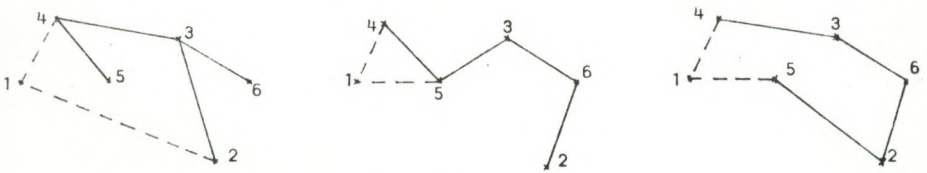
2) A tree on n cities is a set of n-1 edges without any cycle (figure 2).



a tree a tree not a tree

figure 2

Now we give an arbitrary city number 1, without loss of generality. A 1-tree is a tree on all cities but city 1, together with two edges incident to city 1. A minimal 1-tree is a 1-tree with minimal total length (figure 3).



I : a 1-tree II : a minimal 1-tree III : a 1-tree and tour

figure 3

A 1-tree has (n-2) + 2 = n edges and it contains exactly one cycle, because it has one edge more than a tree.

For a given 1-tree the degree of a city is the number of edges incident to this city. In figure 3[I] the degrees are three for city 4, one for city 5 and two for city 2.

Obviously every TSP tour is a 1-tree (with degree = 2 for every city), but not every 1-tree is a tour. Therefore the minimum over all 1-trees is at most equal to the minimum over all TSP tours, and so the length of a minimal 1-tree is a lower bound for the optimal TSP solution.

A minimal 1-tree is easily constructed:

1) Choose an arbitrary city ($\neq 1$) as initial tree.
2) Extend the tree with a city ($\neq 1$), not yet part of the tree, that will increase the length of the tree as little as possible; connect this city with the shortest edge to the tree so far constructed.
3) Repeat rule 2 until all cities are part of the tree.
4) Add the shortest two edges incident to city 1.

In figure 4, city 2 is the initial tree. The shortest extension is 5-2. Then we add successively 3-5, 4-3, and 6-5. We complete the minimal 1-tree with 1-2 and 1-5.

One can prove that this algorithm is correct, see e.g. Dijkstra, A note on two problems in connexion with graphs.

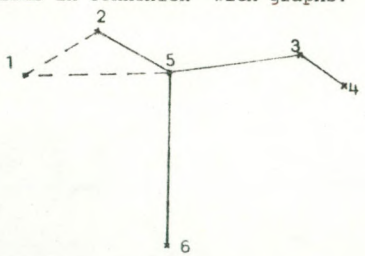


figure 4 : a minimal 1-tree

A minimal 1-tree having degree = 2 for every city is an optimal TSP tour, because there are no shorter 1-trees, and thus no shorter tours.

In the search procedure we calculate lower bounds for the minimum of all tours, containing a given required chain, by constructing a minimal 1-tree on the free cities, where we use the required chain as a generalized city 1.

Weights

So we want a minimal 1-tree that is also a tour. To try for this, we use a simple principle.

Suppose we have n real numbers $\pi_1, \pi_2, \ldots, \pi_n$, with $\pi_1 + \pi_2 + \ldots + \pi_n = 0$. We will call π_i the weight of city i. If we change all distances according to $d'_{ij} = d_{ij} + \pi_i + \pi_j$ (to every distance we add the weights of both related cities), we seem to have a different TSP. But because a tour passes through each city exactly once, every weight will be added twice to the length of the tour. We know that $\pi_1 + \pi_2 + \ldots + \pi_n = 0$, so the length of any tour remains the same, and the shortest tour remains the shortest. However, which 1-tree is minimal depends in general on the weight set, as in the tree the degree of every city is not always two.

In order to obtain a minimal 1-tree with every degree equals two, we give cities with degree = 1 encouragements, and cities with degree >2 penalties by giving them negative, resp. positive weights. We use the formula $\pi_i = c\ (\text{degree}_i - 2)$, $i = 1,2,\ldots,n$, with c a positive constant, so $\pi_1 + \pi_2 + \ldots + \pi_n$ will remain zero. We now hope that the minimal 1-tree looks more like a tour. It will be intuitively clear that better (= higher) lower bounds can be expected when this does occur.

We repeat the process many times, letting c decrease to zero. In most cases this method provides a lower bound close or equal to the optimal TSP value and often an (optimal) tour.

Upper bounds

Values that are definitely larger than the TSP solution are called upper bounds. Obviously the length of any tour is an upper bound, because every tour is at least as long as a shortest one. The method to skip tours during the search process will clearly work better with sharper (= smaller) upper bounds.

Out of every minimal 1-tree, we can easily construct a tour:

1) Consider the unique cycle in the minimal 1-tree.
2) Extend the cycle by connecting a city with degree = 1 directly to the cycle, and by deleting the edge on the cycle just after the connection.
3) Repeat rule 2 until the cycle contains all cities and has become a TSP tour.

For example: In figure 5[a] we start with the cycle 1-5-3-1. After adding 5-4 and deleting 5-3, we get the cycle 1-5-4-6-3-1 (figure 5[b]). After adding 4-2 and deleting 4-6, we get the tour 1-5-4-2-6-3-1.

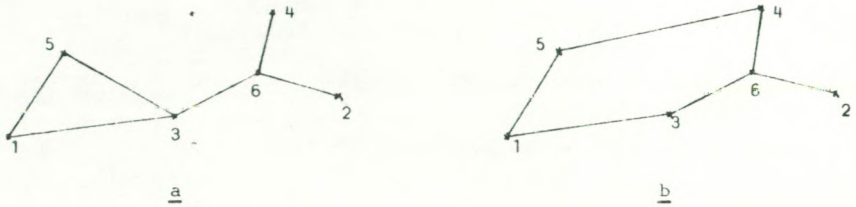


figure 5 : constructing a tour out of a 1-tree

In order to obtain a fast search procedure, it is very important to check "good" tours first, because "bad" tours will then be recognized sooner. Therefore we order our cities according to the tour given by the sharpest upper bound found in the iteration procedure.

An easy check

In figure 6 we have the required chain 1-8-2-5, and a minimal 1-tree. Now we add 5-6 to the required chain. Before calculating a new minimal 1-tree, we get an easy (less sharp) lower bound by adding 5-6 (of course), deleting 5-4 (degree of city 5 must be two), and by replacing 1-3 by 1-4 if $d_{14} < d_{13}$. This is true because the constraint "degree of city 6 is two", has been deleted.

When 1-8-2-5-6 is skipped by this lower bound, we don't have to compute the new minimal 1-tree, saving a lot of time.

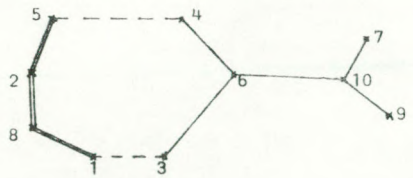


figure 6 : a minimal 1-tree with a required chain

Computational results

We ran the program of Parry and Pfeffer (PPP for short) and our program on an Apple II Europlus computer. Results from a Cyber 750 were used to obtain proper estimates of long execution times on the Apple (this Cyber runs 1200-1600(!) times faster than our Apple).

Parry and Pfeffer's 12-city problem (12 American cities) would have taken at least 1½ hour, solving it by PPP on the Apple, while our program used only 2 minutes 19 seconds. Furthermore we loaded a number of problems with randomly chosen co-ordinates. On four 10-city problems PPP's execution times varied from 1 to 9 hours (!), while our program solved them in 1 to 1½ minute.

Four 15-city problems were solved by our program in 3 to 6 minutes, and some 19- and 20-city problems in 11 to 26 minutes. A well known 33-city problem of Karg and Thompson (33 American cities) was solved by our program in only 53 minutes, while a similar 42-city problem took 5¼ hour.

We did not solve these 15-city and larger problems with PPP, not even on the Cyber, because it should take too much computer time.

In our program we used integer-type variables for those marked in the variable list. 6K Byte was needed to store the program in the Apple II, which means that even a 16K computer can manage problems of about 30 to 40 cities.

Conclusions

The program in this article solves TSP's much faster than that of Parry and Pfeffer. Using a microcomputer, this can save many hours of waiting.

Much more sophisticated and larger programs have been developed to solve TSP's. They are written in higher programming languages and they can deal with larger problems than those mentioned above. For example, Volgenant and Jonker developed a Pascal program that solved a 120-city problem of Grötschel (120 German cities) in 181 seconds on their Cyber.

From this we see that, in spite of the tremendous number of possible tours, very large TSP problems can be solved in a short time.

References

E.W. Dijkstra: A note on two problems in connexion with graphs, Numerische Mathematik 1 (1959) 269-271.

R.L. Karg and G.L. Thompson: A heuristic approach to solving traveling salesman problems, Management Science 10 (1964) 225-248.

R.R. Parry and H. Pfeffer: The Infamous Traveling-Salesman Problem, A Practical Approach, Byte, July 1981, page 252-290.

A. Volgenant and R. Jonker: The symmetric traveling salesman problem and edge exchanges in minimal 1-trees, to appear in European Journal of Operational Research.

Array variables used in the program (*-marked if integer)

A	In a minimal 1-tree computation, A(j) stores the minimal distance from city j to the tree, so far constructed.
*B	In the upper bound computation: $B(j) = \begin{cases} 1, & \text{if city j is part of the unique cycle,} \\ 0, & \text{otherwise.} \end{cases}$
*C	C stores the tour belonging to the sharpest upper bound found in the iteration block.
CH	CH(H) stores the length of the required chain on depth H.
D	D is a 2-dimensional array to store the distance matrix.
*DG	$DG(j) = \begin{cases} -1, & \text{if city j is part of the required chain,} \\ 1+ \text{ degree of city j}, & \text{otherwise.} \end{cases}$
*JS,M,S	Calculating 1-trees, M(j) and S(j) give the minimal and subminimal distances of the last (j=1) resp. the first (j=2) city on the required chain to the tree. IM(j) and JS(j) are the corresponding cities on the tree, for which those minima and subminima are reached.
*K	K(j) stores the number of the $j^{\underline{th}}$ city on the required chain.
N$	N$ is an array to store the names of the cities.
*R	R(j) stores the city on the tree, for which A(j) is minimal. When the computation of the 1-tree is finished, R(j) will be the city, assigned to city j in the 1-tree.
*T	T stores the shortest known tour.
TL	TL is an array for the easy check in the search procedure. In the example (figure 6), TL(4) = length minimal 1-tree - d(5,4) + min{0,d(1,4) - d(1,3)}.
W	W(j) stores the weight for city j.

Most important real and integer (*-marked) variables used in the program

*A,B	Stores the next to last resp. the last city on the required chain.
*BT	Connection point of B with the minimal tree.
*C1,C2	Calculating 1-trees, C2 is the last city added to the tree, and C1 the city that is added next.
*E	Last city on the tour (and first city on the required chain).
F	Used to calculate TL(j). In the example of figure 6 F = d(5,4) - min{0,d(1,4) - d(1,3)}.
*H	Depth in the search procedure (= number of cities on the required chain).
*HE	Used to distinguish between the first and the last city on the required chain.
*I1	After the input block: a variable city number.
*IT	Maximal number of iterations in the iteration block.
*K	K=N, if the starting and ending point are the same; K = N-1, otherwise.
MD	Minimal distance from any city outside the tree to that tree.
MT,ML	Stores the length of a minimal 1-tree, resp. the best lower bound found thus far.
*N	Number of cities specified by the user.
*P1,P2 *P3	Used in the upper bound computation: P1 is a city with degree = 1; P3 is the connection point of the cycle with the path from P1 to the cycle; P2 is the city on the cycle before P3.
*R	R=1, if the optimal TSP value is found in the iteration block; R=0, otherwise.
S	S=CH(H), at the actual depth.
T	Tolerance value for the real numbers to test inequalities.
UT,U	Stores the length of the upper bound, derived from a 1-tree, resp. the best upper bound found thus far.
V	is a precision, specified by the user; two tours of different length will differ at least V.

Listing 1 : The traveling salesman program in Apple Basic.

```
00010 INPUT "HOW MANY DESTINATIONS ";N
00020 IF N>=4 THEN 00040
00030 PRINT "THE NUMBER OF CITIES MUST EXCEED 3" : GOTO 00010
00040 DIM D(N,N),T(N+1),C(N+1),K(N+1),R(N),A(N),CH(N),W(N+1),B(N)
00050 DIM DG(N+1),IM(2),JS(2),M(2),S(2),TL(N),N$(N)
00060 PRINT "INPUT THE NUMBER OF DECIMALS YOU WANT YOUR EDGES TO HAVE ";
00070 PRINT "(0,IF THEY ARE INTEGER,AND MAXIMAL 3).NOTICE THAT MORE ";
00080 PRINT "DECIMALS PROVIDE LONGER EXECUTION TIMES IN GENERAL!"
00090 INPUT G
00100 IF G<>0 AND G<>1 AND G<>2 AND G<>3 THEN 00060
00110 V=10**G : T=0.001/V
00120 REM INPUT INSTRUCTIONS FOR GENERATING DISTANCE-MATRIX.
00130 PRINT "METHOD 1 IF YOU WANT TO INPUT CITIES USING X-Y COORDINATES.
00140 PRINT "METHOD 2 IF YOU WANT TO INPUT A DISTANCE-TABLE."
00150 INPUT "METHOD 1 OR 2? ";I1
00160 IF I1<>1 AND I1<>2 THEN 00150
00170 REM CONSTRUCT INPUT TABLE.
00180 PRINT "YOUR BEGINNING LOCATION IS NUMBER 1"
00190 PRINT "INPUT THE NUMBER OF YOUR ENDING CITY(1 OR ";N;")";
00200 INPUT E
00210 IF E<>1 AND E<>N THEN 00190
00220 K=N-1
00230 IF E=1 THEN K=N
00240 PRINT "INPUT THE NAMES (AND COORDINATES) OF YOUR CITIES "
00250 FOR I=1 TO N
00260 GOSUB 01870
00270 NEXT I
00280 IF I1=2 THEN 00450
00290 REM DISPLAY INPUT TABLE
00300 PRINT
00310 PRINT TAB(15);"INPUT DATA TO BE USED"
00320 PRINT TAB(3);"DESTINATION";TAB(20);"X-COORD.";TAB(30);"Y-COORD."
00330 FOR I=1 TO N
00340 PRINT I;"=";N$(I);TAB(20);CH(I);TAB(30);A(I)
00350 NEXT I
00360 REM EDIT MODE FOR EDITING INPUT DATA
00370 INPUT "DO YOU WANT TO EDIT ANY (Y/N) ";Q$
00380 IF Q$="N" THEN 00460
00390 PRINT:PRINT "TYPE 0 TO END EDITING WHEN ASKED 'WHICH ONE' "
00400 PRINT : INPUT "WHICH ONE  ";I
00410 IF I=0 THEN 00290
00420 IF I<1 OR I>N THEN 00400
00430 GOSUB 01870
00440 GOTO 00400
00450 REM CONSTRUCT INTER-DESTINATION TABLE
00460 IF I1=2 THEN PRINT "CONSTRUCT INTER-DESTINATION TABLE"
00470 FOR I=2 TO N
00480 FOR J=1 TO I-1
00490 IF I1=1 THEN 00520
00500 PRINT "EDGE";I;"-";J;" = ";
00510 INPUT Z : GOTO 00540
00520 X=ABS(CH(I)-CH(J)) : Y=ABS(A(I)-A(J))
00530 Z=(INT(V*(SQR(X*X+Y*Y))+0.5))/V
```

```
00540 D(I,J)=Z : D(J,I)=Z
00550 NEXT J
00560 NEXT I
00570 PRINT
00580 PRINT "DO YOU WANT TO EDIT OR EXAMINE THE DISTANCE-TABLE (Y/N) ";
00590 INPUT Q$
00600 IF Q$="N" THEN 00880
00610 PRINT : PRINT TAB(14);"**** DISTANCE-TABLE ****"
00620 PRINT "(VALUES ROUNDED TO NEAREST INTEGER)"
00630 FOR I=1 TO N
00640 PRINT TAB(4*I);I;
00650 NEXT I
00660 FOR I=1 TO N
00670 PRINT:PRINT I;TAB(3);"!";
00680 FOR J=1 TO N
00690 PRINT INT(D(I,J)+0.5);TAB(4*J+4);
00700 NEXT J
00710 NEXT I
00720 REM EDIT MODE FOR EDITING DISTANCE-TABLE
00730 PRINT:INPUT "DO YOU WANT TO EDIT ANY VALUES (Y/N) ";Q$
00740 IF Q$="N" THEN 00880
00750 PRINT:PRINT "TO ALTER , USE FORMAT:FROM,TO,NEW DISTANCE"
00760 PRINT "FOR EXAMPLE,2,4,512 ALTER THE DISTANCE FROM CITY 2 ";
00770 PRINT "TO CITY 4 TO 512 . DISTANCE FROM CITY 4 TO CITY 2 IS ALSO ";
00780 PRINT "CHANGED . INPUT 0,0,0 TO LEAVE EDIT MODE."
00790 PRINT : PRINT
00800 I=1
00810 PRINT I;" . ";"FROM,TO,DIST= ";
00820 INPUT M,L,D1
00830 IF M=0 THEN 00610
00840 IF M=L OR M>N OR L<1 OR L>N THEN PRINT "ILLEGAL INPUT":GOTO 00810
00850 D(M,L)=D1 : D(L,M)=D1
00860 I=I+1
00870 GOTO 00810
00880 REM  CALCULATE TOTAL POSSIBILITIES FOR TRIP
00890 Z = 1
00900 FOR I=2 TO K-1
00910 Z=Z*I
00920 NEXT I
00930 PRINT "TOTAL POSSIBILITIES FOR TRIP ";Z
00940 C(1)=1 : C(K+1)=E
00950 V=1/V-T
00960 M=32000:U=M:ML=-M
00970 REM TO FIND SHARP LOWER BOUNDS . START ITERATION PROCEDURE
00980 PRINT "THE ITERATION PROCEDURE IS AT WORK AT THE MOMENT"
00990 B=1:H=1:R=0
01000 IT=3*K
01010 FOR L=1 TO IT
01020 GOSUB 01980
01030 IF MT>U-V THEN R=1:L=IT+1:GOTO 01130
01040 IF MT>ML+T THEN ML=MT
01050 GOSUB 02370
01060 IF UT>U-T THEN 01110
```

```
01070 U=UT : I1=1
01080 FOR I=2 TO K
01090 I1=R(I1) : C(I)=I1
01100 NEXT I
01110 IF ML>U-V THEN R=1:L=IT+1:GOTO 01130
01120 IF L<>IT THEN GOSUB 01740
01130 NEXT L
01140 FOR I=1 TO N
01150 T(I)=I : K(I)=1
01160 NEXT I
01170 IF R=1 THEN 01660
01180 REM CALCULATE SHORTEST TRIP IN THE SEARCH PROCEDURE
01190 PRINT "THE SEARCH PROCEDURE IS AT WORK AT THE MOMENT"
01200 TL(1)=MT-F
01210 REM RESEQUENCE THE CITIES
01220 FOR I=2 TO N
01230 FOR J=1 TO I-1
01240 A=C(I):B=C(J)
01250 IF A>B THEN EX=A:A=B:B=EX
01260 D(I,J)=D(A,B)
01270 NEXT J
01280 NEXT I
01290 FOR I=1 TO N-1
01300 FOR J=I+1 TO N
01310 D(I,J)=D(J,I)
01320 NEXT J
01330 NEXT I
01340 CH(1)=0:B=1
01350 REM SEARCH PROCEDURE
01360 A=B : B=1 : H=H+1
01370 DG(B)=1
01380 IF B=K THEN 01440
01390 D=B
01400 FOR I=B+1 TO K
01410 IF DG(I)>0 THEN B=I:K(H)=B:DG(B)=-1:I=K+1
01420 NEXT I
01430 IF D<>B THEN 01480
01440 H=H-1
01450 IF H=1 THEN 01660
01460 B=A : A=K(H-1) : GOTO 01370
01470 REM GENERAL CHECK
01480 S=CH(H-1)+D(A,B) : CH(H)=S
01490 IF S+TL(H-1)>U-V THEN 01370
01500 GOSUB 01980
01510 TL(H)=MT-F : MT=MT+S
01520 IF MT>U-V THEN 01370
01530 GOSUB 02370
01540 IF UT<U-T THEN U=UT
01550 IF MT<U-V THEN 01360
01560 REM STORE THE BETTER TOUR
01570 FOR I=2 TO H
01580 T(I)=K(I)
01590 NEXT I
```

```
01600 I1=B
01610 FOR I=H+1 TO K
01620 I1=R(I1) : T(I)=I1
01630 NEXT I
01640 GOTO 01370
01650 REM OPTIMAL TOUR FOUND , DISPLAY RESULTS.
01660 PRINT:PRINT "OPTIMAL TOUR :"
01670 FOR I=1 TO K
01680 PRINT I,N$(C(T(I)))
01690 NEXT I
01700 PRINT K+1,N$(E)
01710 PRINT "THE LENGTH OF THE SHORTEST TRIP IS ";U
01720 GOTO 02550
01730 REM SUBROUTINE 1 TO CHANGE DISTANCE MATRIX BY WEIGHTS
01740 Z=(IT-L+1)*(IT-L+1)*MT/IT/IT/50
01750 FOR I=2 TO K
01760 W(I)=Z*(DG(I)-3)
01770 NEXT I
01780 W(1)=0 : W(K+1)=0
01790 FOR I=1 TO N-1
01800 X=W(I)
01810 FOR J=I+1 TO N
01820 Y=W(J) : Z=D(I,J)+X+Y
01830 D(I,J)=Z : D(J,I)=Z
01840 NEXT J
01850 NEXT I
01860 RETURN
01870 REM SUBROUTINE 2 TO INPUT CITY NAME (AND COORDINATES)
01880 PRINT:PRINT I;".";
01890 PRINT TAB(5);"NAME OF CITY ";
01900 INPUT N$(I)
01910 IF I1=2 THEN RETURN
01920 PRINT TAB(5);"X-COORDINATE ";
01930 INPUT CH(I)
01940 PRINT TAB(5);"Y-COORDINATE ";
01950 INPUT A(I)
01960 RETURN
01970 REM SUBROUTINE 3 TO CALCULATE A 1-TREE
01980 FOR I=2 TO K
01990 IF DG(I)>=0 THEN DG(I)=1:A(I)=M
02000 NEXT I
02010 HE=B
02020 FOR J=1 TO 2
02030 M(J)=M : S(J)=M
02040 FOR I=2 TO K
02050 IF DG(I)<0 THEN 02100
02060 Z=D(HE,I)
02070 IF Z>S(J)-T THEN 02100
02080 IF Z>M(J)-T THEN S(J)=Z:JS(J)=I:GOTO 02100
02090 S(J)=M(J):JS(J)=IM(J):M(J)=Z:IM(J)=I
02100 NEXT I
02110 IF B=E THEN M(2)=S(1):IM(2)=JS(1):J=3
02120 HE=E
```

```
02130 NEXT J
02140 IF IM(1)<>IM(2) THEN 02180
02150 HE=2
02160 IF M(1)+S(2)>M(2)+S(1)+2*T THEN HE=1
02170 M(HE)=S(HE) : IM(HE)=JS(HE)
02180 R(B)=IM(1):C2=IM(2):R(C2)=E:DG(C2)=2
02190 MT=M(1)+M(2) : BT=R(B)
02200 F=D(E,C2)-D(E,BT)
02210 IF F<0 THEN F=0
02220 F=F+D(B,BT)
02230 FOR I=H+2 TO K
02240 MD=M
02250 FOR J=2 TO K
02260 IF DG(J)<>1 THEN 02300
02270 Z=D(J,C2)
02280 IF Z<A(J)-T THEN A(J)=Z:R(J)=C2
02290 IF A(J)<MD-T THEN MD=A(J):C1=J
02300 NEXT J
02310 C2=C1:MT=MT+MD:DG(C2)=2
02320 DG(R(C2))=DG(R(C2))+1
02330 NEXT I
02340 DG(R(B))=DG(R(B))+1
02350 RETURN
02360 REM SUBROUTINE 4 TO CALCULATE AN UPPER BOUND OUT OF A 1-TREE
02370 FOR I=2 TO K
02380 B(I)=0
02390 NEXT I
02400 UT=MT : I1=R(B)
02410 B(I1)=1 : I1=R(I1)
02420 IF I1<>E THEN 02410
02430 FOR I=2 TO K
02440 IF DG(I)<>2 THEN 02530
02450 P1=I:B(P1)=1:P3=R(P1)
02460 IF B(P3)=0 THEN B(P3)=1:P3=R(P3):GOTO 02460
02470 I1=B
02480 P2=I1 : I1=R(I1)
02490 IF I1<>P3 THEN 02480
02500 Z=D(P1,P2)-D(P3,P2)
02510 R(P2)=P1 : UT=UT+Z
02520 IF UT>U-T THEN RETURN
02530 NEXT I
02540 RETURN
02550 END
```

Listing 2 : Run of the traveling salesman program on the 12-city problem of Parry and Pfeffer. The execution time is 2 minutes 19 secs on an Apple II .

```
HOW MANY DESTINATIONS  ? 12

INPUT THE NUMBER OF DECIMALS YOU WANT YOUR EDGES TO HAVE
(0,IF THEY ARE INTEGER,AND MAXIMAL 3).NOTICE THAT MORE
DECIMALS PROVIDE LONGER EXECUTION TIMES IN GENERAL'
?1

METHOD 1 IF YOU WANT TO INPUT CITIES USING X-Y COORDINATES
METHOD 2 IF YOU WANT TO INPUT A DISTANCE-TABLE.
METHOD 1 OR 2  ? 1

YOUR BEGINNING LOCATION IS NUMBER 1
INPUT THE NUMBER OF YOUR ENDING CITY(1 OR 12 ) ? 1

INPUT THE NAMES (AND COORDINATES) OF YOUR CITIES

 1 . NAME OF CITY ?peoria

     X-COORDINATE ?-93.6

     Y-COORDINATE ?-87.3


 2 . NAME OF CITY ?chicago

     X-COORDINATE ?0

     Y-COORDINATE ?0


 3 . NAME OF CITY ?belleville

     X-COORDINATE ?-114.4

     Y-COORDINATE ?-234.6


 4 . NAME OF CITY ?carbondale

     X-COORDINATE ?-76.9

     Y-COORDINATE ?-286.9


 5 . NAME OF CITY ?rockford
```

```
X-COORDINATE ?-66.9

Y-COORDINATE ?20.5

6 . NAME OF CITY ?decatur

X-COORDINATE ?-61.7

Y-COORDINATE ?-145.4

7 . NAME OF CITY ?waukegen

X-COORDINATE ?-6.5

Y-COORDINATE ?26.2

8 . NAME OF CITY ?champaign

X-COORDINATE ?-19.7

Y-COORDINATE ?-124.4

9 . NAME OF CITY ?dekalb

X-COORDINATE ?-57.9

Y-COORDINATE ?-4.0

10 .NAME OF CITY ?springfield

X-COORDINATE ?-94.3

Y-COORDINATE ?-151.0

11 .NAME OF CITY ?kankakee

X-COORDINATE ?-4.1

Y-COORDINATE ?-58.9
```

```
12 .NAME OF CITY ?aurora

     X-COORDINATE ?-31.1

     Y-COORDINATE ?-13.8


                INPUT DATA TO BE USED
   DESTINATION        X-COORD.   Y-COORD.
 1 .PEORIA            -93.6      -87.3
 2 .CHICAGO            0          0
 3 .BELLEVILLE        -114.4     -234.6
 4 .CARBONDALE        -76.9      -286.9
 5 .ROCKFORD          -66.9       20.5
 6 .DECATUR           -61.7      -145.4
 7 .WAUKEGEN          -6.5        26.2
 8 .CHAMPAIGN         -19.7      -124.4
 9 .DEKALB            -57.9      -4
 10 .SPRINGFIELD      -94.3      -151
 11 .KANKAKEE         -4.1       -58.9
 12 .AURORA           -31.1      -13.8
DO YOU WANT TO EDIT ANY (Y/N)  ? y


TYPE 0 TO END EDITING WHEN ASKED 'WHICH ONE'
WHICH ONE   ?7


 7 . NAME OF CITY ?waukegan

     X-COORDINATE ?-6.5

     Y-COORDINATE ?26.2

WHICH ONE   ?0


                INPUT DATA TO BE USED
   DESTINATION        X-COORD.   Y-COORD.
 1 .PEORIA            -93.6      -87.3
 2 .CHICAGO            0          0
 3 .BELLEVILLE        -114.4     -234.6
 4 .CARBONDALE        -76.9      -286.9
 5 .ROCKFORD          -66.9       20.5
 6 .DECATUR           -61.7      -145.4
 7 .WAUKEGAN          -6.5        26.2
 8 .CHAMPAIGN         -19.7      -124.4
 9 .DEKALB            -57.9      -4
 10 .SPRINGFIELD      -94.3      -151
 11 .KANKAKEE         -4.1       -58.9
 12 .AURORA           -31.1      -13.8
DO YOU WANT TO EDIT ANY (Y/N)  ? n
```

```
DO YOU WANT TO EDIT OR EXAMINE THE DISTANCE-TABLE (Y/N)  ? n
TOTAL POSSIBILITIES FOR TRIP  39916800
THE ITERATION PROCEDURE IS AT WORK AT THE MOMENT

OPTIMAL TOUR :
 1              PEORIA
 2              SPRINGFIELD
 3              BELLEVILLE
 4              CARBONDALE
 5              DECATUR
 6              CHAMPAIGN
 7              KANKAKEE
 8              AURORA
 9              CHICAGO
 10             WAUKEGAN
 11             ROCKFORD
 12             DEKALB
 13             PEORIA
THE LENGTH OF THE SHORTEST TRIP IS  761.7
```

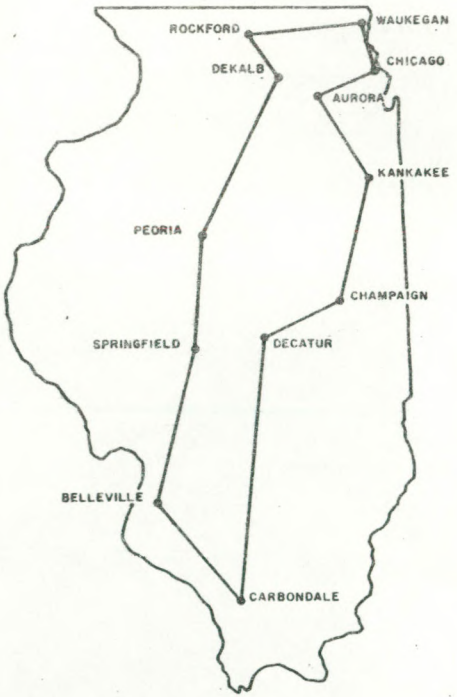


figure 7 : Optimal tour for Parry and Pfeffer's 12-city problem