Mathematical Programming 10 (1976) 367–378.
North-Holland Publishing Company

INTEGER PROGRAMMING APPROACHES TO THE TRAVELLING SALESMAN PROBLEM

P. MILIOTIS
University of London, London, U.K.

Received 30 July 1974
Revised manuscript received 25 July 1975

The availability of an LP routine where we can add constraints and reoptimize, makes it possible to adopt an integer programming approach to the travelling-salesman problem.

Starting with some of the constraints that define the problem we use either a branching process or a cutting planes routine to eliminate fractional solutions. We then test the resulting integer solution against feasibility and if necessary we generate the violated constraints and reoptimize until a "genuine" feasible solution is achieved.

Usually only a small number of the omitted constraints is generated.

The generality of the method and the modest solution times achieved leads us to believe that such an LP approach to other combinatorial problems deserves further consideration.

1. Introduction

The Travelling Salesman Problem can be described as follows: Find the order in which a salesman must visit each one of a set of n cities and return to the starting city so that the total distance travelled is minimized. In graph theoretic terminology this may be stated as: find the shortest Hamiltonian circuit of a graph.

Integer programming formulations of the problem appeared in [3,4, 15] and were reviewed in [2].

The one that is used in this paper is repeated below:

$$\begin{aligned} &\text{minimize} && \sum_{i,j} x_{ij}\, c_{ij}\ , \\ &\text{subject to} && \sum_{j} x_{ij} = 1 \quad (i = 1, \ldots, n)\ , \end{aligned} \tag{1.1}$$

$$\sum_i x_{ij} = 1 \quad (j = 1, \ldots, n),$$

$$\sum_{\substack{i \in S \\ j \in \bar{S}}} x_{ij} \geqslant 1 \tag{1.2}$$

for all partitions $(S, \bar{S})$ of the set of n cities such that $2 \leqslant |S| \leqslant n - 2$ where by $|S|$ we denote the number of elements in S.

$$x_{ij} = 0, 1 . \tag{1.3}$$

In the symmetric case (i.e. when $c_{ij} = c_{ji}$) the above set of constraints is reduced to:

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \quad (k = 1, \ldots, n), \tag{1.4}$$

$$\sum_{\substack{i \in S \\ j \in \bar{S}}} x_{ij} \geqslant 2 , \tag{1.5}$$

$$x_{ij} = 0, 1. \tag{1.6}$$

In this case only variables x_{ij} with $i < j$ are considered (or more generally if x_{ij} is considered to be a variable x_{ji} is not). Also if partition $(S, \bar{S})$ is considered, partition $(\bar{S}, S)$ is not and S must be such that $3 \leqslant |S| \leqslant n - 3$.

Thus it should be clear that the symmetric case involves half the number of variables and constraints than the asymmetric and no loops of length 2. Perhaps it is worth mentioning that whenever the set of constraints (1.4) is satisfied the set of constraints (1.5) is equivalent to:

$$\sum_{\substack{i \in S \\ j \in S}} x_{ij} \leqslant |S| - 1 . \tag{1.7}$$

Constraints (1.1) or (1.4) describe the related assingment problem.

Constraints (1.2) or (1.5) are called the loop constraints because they describe the proper connectedness of the graph that represents a given integer solution or the equivalent condition of blocking subtours (form (1.7)).

A number of branch and bound algorithms [10,13,17] have been

developed for this class of problem. The brief review that follows will help to point out the differences with the presently proposed method.

In the Little (et al.) [13] algorithm, at every stage a variable is chosen and fixed at its two alternative values thus separating the current subset of feasible solutions into two other subsets. The assignment problem which is used to evaluate a lower bound for each of the two subsets is not actually solved but a lower bound to the assignment problem is obtained (which is also a lower bound to each of the two subsets). This results in a relatively big tree with relatively little computational effort at each node. The separation process continues until either a feasible solution is discovered, or the current subset is fathomed [7].

Loop constraints are treated implicitly by fixing at 0 after every separation stage the variables that might create subloops.

In Eastman's [5] and Shapiro's [17] algorithms the related assignment problem is solved at every stage and whenever subloops occur one of them, say of length k, is chosen and a multiple branch into k subproblems is made restricting in turn each of the variables that form the subloop to take a zero value. This way of eliminating the subloops is somewhat refined in [1], which also reports using Edmonds' and Johnson's 2-matching algorithm for subproblem solution [6].

Finally in [10] the minimum spanning tree is used to derive a tight bound thus creating a compact tree at the expense of heavier computation at each node.

A common characteristic in the above algorithms is that they do not allow fractional values of the variables to occur. By choosing to deal either with the assignment problem or the minimum spanning tree they never generate fractional solutions and the subproblems can be solved by using a more efficient routine than the simplex method.

The algorithm that we will present treats both the assingment constraints and the loop constraints explicitly thus allowing the occurrence of fractional solutions which are then eliminated by branching.

In that sense it is more similar to the earlier methods proposed for the solution of the problem by Dantzig, Fulkerson and Johnson [3,4], and particularly to that proposed by Martin [14].

2. The algorithm

The terminology used to describe the following algorithm is derived from Geoffrion and Marsten [7].

The basic feature of the algorithm is that is uses two types of relaxation.

The first type (R1), is the usual relaxation of the integrality conditions. As a result fractional solutions may arise. Using Little's principle [13] we choose one of the variables currently not satisfying the integrality requirements and fix it alternatively at each of its two possible values thereby separating the current subproblem into two others (descendants), one of which becomes the current subproblem to be analysed, the other entering the candidate list.

The second type of relaxation (R2) consists of starting the problem with a subset of the set of constraints that define it.

Let C be the set of the constraints that completely describe the feasible solutions to the problem (apart from the integrality conditions) and (C_1, C_2) be a partition of C.

Initially a solution is found satisfying the constraints that belong to the set C_1. Some of the constraints that belong to C_2 may then be satisfied by coincidence. But generally in the absence of these constraints we may get integer solutions – in most cases after having branched sufficiently – which will be infeasible because they violate some of the omitted constraints. We then generate some (or all) of the violated constraints and reoptimize the current candidate problem.

This is where the present approach is very similar to the approach taken in [3,4], but it is more general in three ways.

In the first place the relaxation can be used within a branch and bound context.

Secondly, in principle any subset C_1 of the original set of constraints can provide a valid starting point.

Thirdly, the additional constraints are generated automatically by a single computer run. This is made possible by the use of the Land–Powell routines [12]. These enable subroutines to be added to the programs to test the genuine feasibility of an integer solution, to add any violated constraints, and a return to feasibility and integrality to be made.

The use of such a method may be justified by the following considerations.

(1) Integer feasible solutions to some combinatorial problems have a specified structure which can be asserted without it being necessary to explicitly test that all the constraints defining that structure are satisfied.

In the case of the travelling–salesman problem, using a proper labelling of the cities (nodes of the graph), not only are we able to examine whether an integer solution is a Hamiltonian cycle, but we also have all the information to generate the violated constraints if it is not.

(2) By generating only the necessary constraints (i.e. those that are violated by the current integer solution) we may arrive at the optimal solution and prove it to be optimal (by exploiting a sufficient number of branches) having used only a very small proportion of the original (total) number of constraints. The rest of the constraints will remain ineffective throughout the process and therefore they will never be generated explicitly.

In what follows, the basic steps of the algorithm are described. A Flowchart is given in Fig. 1, and some explanations follow. The capitalized names used in the steps refer either to the names of subroutines or to values of the parameters in the programs used. Since the program used is set to maximize a function, the objective function is of the form $\max\{-\Sigma c_{ij} x_{ij}\}$.

Step 1: Initialization: Define the arcs that form the graph representing the problem. Generate the constraints that constitute C_1. Define the initial values for the parameters of the LP and the Branch and Bound routine. e.g. initialize the candidate list to contain the initial LP problem, the value of the best feasible solution discovered so far (BEST) to be a large negative number etc. (Subroutines BBDATA and DATA).

Step 2: Select a problem from the candidate list to become the current subproblem.

Step 3: Optimize (or reoptimize) the current subproblem (subroutine LP).

Step 4: Fathoming Criteria (subroutine ISTAIL).

4.1. If the current subproblems is infeasible go to step 10 (ITAIL = 3).

4.2. If the LP-optimal value of the current subproblem is not better than BEST, go to step 10 (ITAIL = 2).

4.3. If the LP-optimal value of the current subproblem is not integer go to step 9. (ITAIL = 0).

Step 5: The current solution is integer. (ITAIL = 1). Label the nodes of the graph that represents the optimal solution of the current subproblem (subroutine VERTEX).

Step 6: If all the assignment constraints are satisfied go to step 7. This is indicated by MHOLD still being equal to MNOW (M now) the current

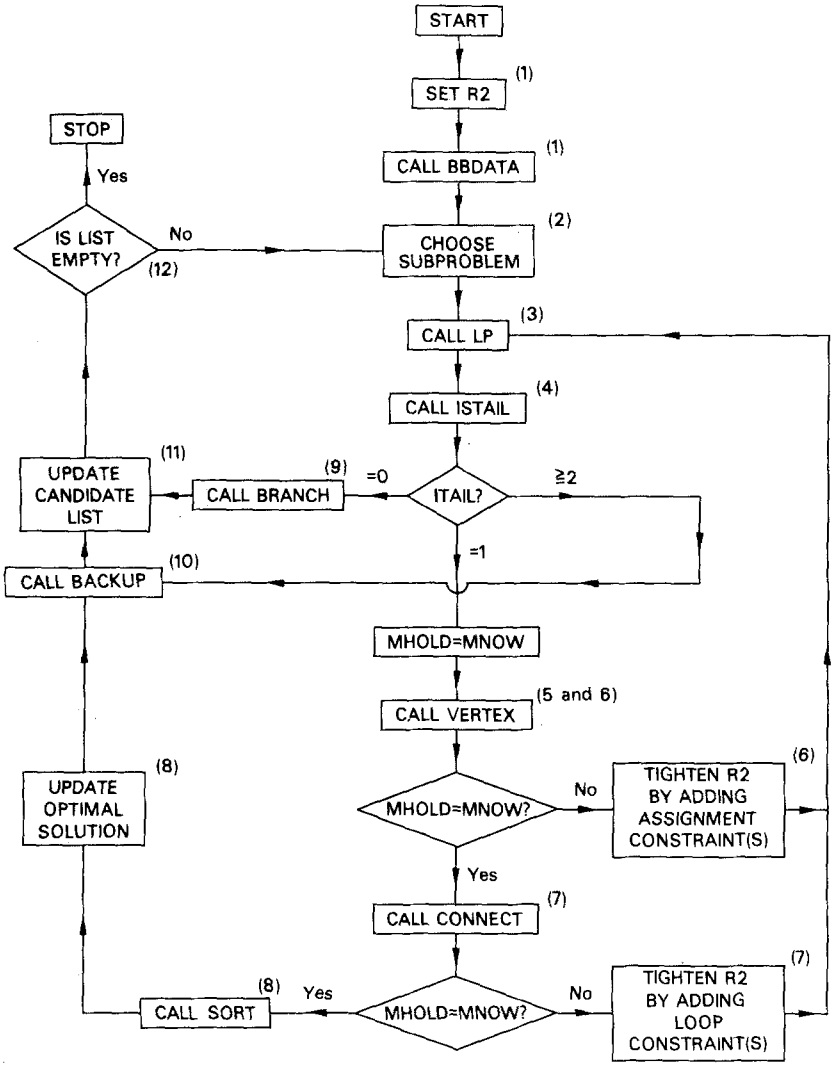

Fig. 1.

number of constraints in C_1. Otherwise generate one (or more) of the violated vertex constraints and go to step 3 (subroutine VERTEX).

Step 7: If all the loop constraints are satisfied (this is tested again by comparing MHOLD and MNOW) go to step 8. Otherwise generate some (or all) of the violated loop constraints and go to step 3 (subroutine CONNECT).

Step 8: A feasible solution with value better than BEST has been found. Call subroutine SORT to get the order of the visit of the cities for that solution, update BEST and go to step 10.

Step 9: Separate the current subproblem into two other subproblems and go to step 11 (subroutine BRANCH).

Step 10: Backtrack (subroutine BACKUP).

Step 11: Update candidate list.

Step 12: Check if candidate list is empty. If it is empty stop otherwise and go to step 2.

Remarks. With the exception of VERTEX and CONNECT which are briefly described below, these are the subroutines described in [12], and used here with minor modifications.

Subroutine VERTEX sets up a proper labelling of the nodes which enables us to test feasibility. The rules used are the following:

(1) The nodes of the graph that belong to different disconnected parts will have labels of different absolute values and the nodes of the graph that belong to a connected part will have labels of the same absolute value.

(2) The nodes that have an odd number of arcs incident to them will have a negative label and the nodes that have an even number of arcs incident to them will have a positive label. VERTEX also generates explicitly any assignment constraints that are violated at the current integer point.

Subroutine CONNECT generates any violated loop constraints at the current integer point using the labelling set up in VERTEX. CONNECT is entered only if the assignment constraints are satisfied and if no constraints are generated in CONNECT the solution is feasible.

Both VERTEX and CONNECT can only be entered if an integer solution (feasible or not) has been found.

It should be clear that the added constraints in VERTEX and CONNECT are valid throughout the problem (and not only for the descendants of the current subproblem as in the case of adding cutting planes) and that one must take proper account of the variables already fixed by branching.

Finally, in the actual program the updating of the candidate list is embedded within the subroutines BRANCH and BACKUP and the test whether the list is empty in BACKUP. In the Flowchart they are presented as different steps in order to point out the connection with a general branch and bound procedure.

3. Computational results

Table 1 shows some of the bigger problems that were tried of those found in the literature using the branch and bound algorithm. The computer used was the CDC 7600. The last two columns of the table show the results reported in [10] and in [8] respectively, for the same prob-

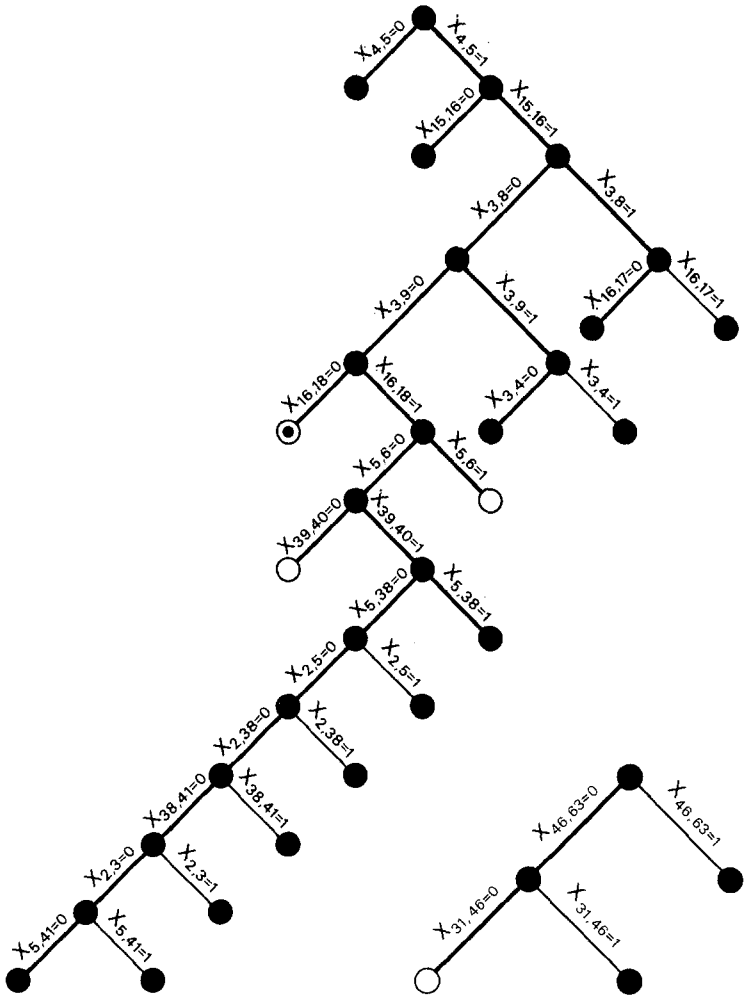

Fig. 2. 42 cities (Dantzig)

Fig. 3. 8 X 8 Knight's tour.

○ feasible solution, ◉ optimal solution

Table 1
Branch-and-bound algorithm problems from earlier papers

Problem	Source	n (no. of cities)	No. of variables	Generated loop constraints	Optimal value	LP iterations	Max. size of the inverse	Time (sec)	Time (sec) reported in [10]	Time (sec) in [8]
1	Dantzig et al. [3]	42	861	12	699	274	49	14.5	54.0	9.8
2	Dantzig et al. reduced [3,16]	42	353	16	12345	698	49	22.2		
3	Held & Karp [9]	48	1128	17	11461	199	58	10.4	84.0	9.8
4	Held & Karp reduced [10,16]	48	790	17	11461	206	58	7.5		
5	Karg & Thompson [11]	57	1596	26	12955	1031	69	128.3	780.0	134.0
6	Karg & Thompson reduced [11,16]	57	473	34	12955	2036	68	100.0		
7	Tutte [10]	46	69	42	[a])	674	54	11.1	900.0	591.0
8	3 × 10 Knight's tour [10]	30	50	5	0	95	32	0.33	0	1.1
9	6 × 8 Knight's tour [10]	48	116	13	0	335	50	3.28	15.96	10.0
10	8 × 8 Knight's	64	168	4	0	424	64	3.9	415.8	38.0

[a]) No solution.

lems (wherever applicable). In the former case an IBM 360/91 was used and in the latter case an IBM 360/75.

The comparative speed of the different computers involved in the above computations is very hard to estimate accurately. According to certain information the CDC 7600 is approximately 10 times faster than the IBM 360/75 [18, p. 165] and the IBM 360/91 3 times faster than the IBM 360/75 [8, p. 93].

The starting set of constraints C_1 was the set of the assignment constraints. Figs. 2 and 3 show the resulting tree graphs of the branch and bound procedure for problems (1) and (10).

Heavy lines represent the branches that were actually explored and light lines branches that were fathomed due to their bounds.

All the problems derived from road networks were run twice. The first time using the complete distance matrix and the second time using a reduced distance matrix by a procedure suggested in [16]. The reduction takes place in two steps. Firstly using Floyd's algorithm we derive the shortest distances say c'_{ij}. Then running Floyd's algorithm backwards we drop all the variables x_{ij} for which $c'_{ij} = c'_{ik} + c'_{kj}$ for some k. This of course may result in cutting off the optimal route or even producing a graph with no Hamiltonian circuit.

However in the case where we allow visiting each city at least once (rather than exactly once) this reduction will still necessarily give the optimal solution, but this case is not examined here.

In creating problem (2) we tried to restore the original distance matrix by multiplying each cost by 17 and adding 11 to the product. Since the costs appearing in [3, p. 395] were rounded off, the resulting distance matrix for problem (2) may well be approximate. Note however that $699 \times 17 + 42 \times 11 = 12345$.

Time includes input and output of data and results.

In problem (8) the solution was obtained without branching.

Some recent work, which will be reported in another paper, has used a cutting plane algorithm instead of a branch and bound to achieve integrality. It appears to be more efficient except for cases in which there is no feasible solution.

A modified version of the programs solves asymmetric problems but the computational results are not reported here.

4. Conclusion

The modest solution times achieved for problems which are recognized as testing ones give us encouragement to believe that such an LP approach to combinatorial problems deserves further consideration. All that is needed is a routine to test the genuine feasibility of an integer solution, and if it is not feasible to turn on a device for generating the violated constraints.

Finally in each case one may experiment using different branching rules and different starting points (partition into C_1 and C_2).

Acknowledgment

I would like to thank Dr. A. Land, the supervisor of my research, for her guidance and support. However, I should be held solely responsible for any errors or omissions that might occur in the present paper.

References

[1] M. Bellmore and J.C. Malone, "Pathology of travelling salesman subtour elimination algorithms", *Operations Research* 19 (1971) 278–307.

[2] M. Bellmore and G.L. Nemhauser, "The travelling salesman problem: a survey", *Operations Research* 16 (1968) 538–558.

[3] G.B. Dantzig, D.R. Fulkerson and S.M. Johnson, "Solution of a large scale travelling salesman problem", *Operations Research* 2 (1954) 393–410.

[4] G.B. Dantzig, D.R. Fulkerson and S.M. Johnson, "On a linear programming, combinatorial approach to the travelling salesman problem", *Operations Research* 7 (1959) 58–66.

[5] W.L. Eastman, "Linear programming with pattern constraints", Ph.D. Dissertation, Harvard University, Cambridge, Mass., (1958).

[6] J. Edmonds, "Maximum matching and a polyhedron with 0, 1-vertices", *Journal of Research of the National Bureau of Standards* 69B (1965) 125–130.

[7] A.M. Geoffrion and R.E. Marsten, "Integer programming algorithms: A framework and state-of-the-art survey", *Management Science* 18 (1972) 465–491.

[8] K. Helbig Hansen ahd J. Krarup, "Improvements of the Held–Karp algorithm for the symmetric travelling-salesman problem", *Mathematical Programming* 7 (1974) 87–96.

[9] M. Held and R.M. Karp, "A dynamic programming approach to sequencing problems", *Journal of the Society for Industrial and Applied Mathematics* 10 (1962) 196–210.

[10] M. Held and R.M. Karp, "The travelling salesman problem and minimum spanning trees, Part II", *Mathematical Programming* 1 (1971) 6–25.

[11] L.L. Karg and G.L. Thompson, "A heuristic approach to solving travelling salesman problems", *Management Science* 10 (1964) 225–248.

[12] A. Land and S. Powell, *Fortran codes for mathematical programming* (Wiley, New York, 1973).
[13] J.D. Little, K.G. Murty, D.W. Sweeney and C. Karel, "An algorithm for the travelling salesman problem", *Operations Research* 11 (1963) 972–989.
[14] G.T. Martin, "Solving the travelling salesman problem by integer linear programming", *CEIR*, New York (1966).
[15] C.E. Miller, A.W. Tucker and R.A. Zemlin, "Integer programming formulations and travelling salesman problems", *Journal of the Association for Computing Machinery* 7 (1960) 326–329.
[16] J.D. Murchland, "A fixed matrix method for all shortest distances in a directed graph and for the inverse problem", Ph.D. Dissertation, Karlsruhe (1970).
[17] D. Shapiro, "Algorithms for the solution of optimal cost travelling salesman problem", Sc.D. Thesis, Washington University, St. Louis, Mo. (1966).
[18] "Computers in Central Government: Ten Years Ahead", Civil Service Department, *Management Studies* 2, HMSO London (1971) (no authors reported).