EURO Journal on Computational Optimization 9 (2021) 100017

Contents lists available at ScienceDirect

EURO Journal on Computational Optimization

journal homepage: www.elsevier.com/locate/ejco

The Solution of some 100-city Travelling Salesman Problems

A. Land[1]

London School of Economics, United Kingdom, 1979

A R T I C L E   I N F O

Keywords:
Travelling salesman problem
Cut-and-price
Combinatorial optimization

A B S T R A C T

A simplex-based FORTRAN code, working entirely in integer arithmetic, has been developed for the exact solution of travelling-salesman problems. The code adds tour-barring constraints as they are found to be violated. It deals with fractional solutions by adding two-matching constraints and as a last resort by 'Gomory' cutting plane constraints of the Method of Integer Forms. Most of the calculations are carried out on only a subset of the variables, with only occasional passes through the whole set of possible variables. Computational experience on some 100-city problems is reported.

The solution of symmetric travelling salesman problems of orders up to 80 by the automatic application of Gomory's cutting planes to an LP formulation was reported by Miliotis [7]. This used an all-in-core Fortran exact arithmetic code. The size limit of 80 was set by the amount of storage required as the number of variables increased, and (in later work than that reported by Miliotis) by the size of the determinant becoming too large for an exact arithmetic code. This paper reports further work to remove these limitations.

The symmetric travelling salesman problem (TSP) on n cities can be expressed as an integer linear program (ILP) on the variables x_{ij}, $i < j$, $i = 1, \dots, n-1$; $j = i+1, \dots, n$:

$$\text{Minimize} \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$

$$\text{subject to} \quad x_{ij} \geq 0 \tag{1}$$

$$x_{ij} \leq 1 \tag{2}$$

$$\sum_{i=1}^{k-1} x_{ik} + \sum_{j=k+1}^{n} x_{kj} = 2 \tag{3}$$

$$\sum_{i\in S, j\in S} x_{ij} \leq |S| - 1 \tag{4}$$

for every set S of vertices

for $2 < |S| < n$

$$x_{ij} \text{ integer} \tag{5}$$

Constraints (3) are referred to as the 'vertex constraints' and ensure that every vertex (city) is connected by edges with a total weight of exactly two to the rest of the graph.

Constraints (4) are 'subtour-barring' constraints and (together with constraints (1), (2) and (3)) ensure that every strict subset of the n vertices is connected to the rest of the graph by edges with a total weight of at least two.

Constraints (5) are dealt with by the addition of 'two-matching' constraints [1] and by constraints of Gomory's Method of Integer Forms [2].

The algorithm can be regarded as a further development of Miliotis' 'reverse' algorithm, in that no attempt is made to deal with constraints (5), integrality of the variables, unless no subtour-barring constraints can be found (not merely those which are associated with a disconnected subset of vertices). The storage limitation has been stretched by solving the problem only on a subset of the variables and then using a column generation routine to examine, and if necessary to introduce, additional variables. The problem of the increasing size of the determinant has been overcome by a development in the cutting plane algorithm.

E-mail address: immanuel.bomze@univie.ac.at

[1] Ailsa H. Land sadly passed away on 16 May 2021; for an obituary see [9] which needs a slight correction: in contrast to the statement in [9], the EURO Gold Medal was, according to the guidelines, awarded to Ailsa Land before 16 May 2021, not posthumously. This unpublished typewritten manuscript with her handwritten annotations was typeset anew. Any editorial remarks are contained in footnotes, and the last two references [8,9] added by the Editor-in-Chief in compliance with the original citation style. Publication of this historic document in EJCO was kindly authorized by Professor Frank Land, LSE, the widower of A. Land. The Editor-in-Chief is indebted to Professors Gilbert Laporte and Panagiotis Miliotis for pointing out the existence of this manuscript as well as for making a complete version of it readily available.

https://doi.org/10.1016/j.ejco.2021.100017
Received 27 August 2021; Accepted 18 October 2021

Outline of the algorithm

The algorithm is described by the flow diagram,[2] Diagram 1, and the following outline description.

Step 1

A heuristic algorithm is used to provide an initial complete tour solution, an "incumbent", the best tour solution so far. A selection of n_k 'active' variables is made from the total possible set of $n(n-1)/2$ variables. The selection includes the variables entering the heuristic tour.

Step 2

An LP solution to the problem with the current set of variables and constraints (initially with constraints of types (1), (2) and (3) only) is obtained. Here and elsewhere all LP calculations are carried out in exact arithmetic.

Step 3

If the optimum solution value is such that it is clear that there cannot be a lower cost solution using the current set of active variables than the incumbent, go to Step 19. Otherwise:

Step 4

The vertices are labelled into subsets connected by edges of weight at least one half, and the total weight of edges internal to each subset is accumulated.

Step 5

If the edge weight within at least one subset is greater than $|S|-1$, a constraint for **each** labelled subset is added to the LP, the current solution of which is therefore infeasible, and the algorithm returns to Step 2. Otherwise:

Step 6

If Step 5 has failed to find a violated tour-barring constraint **AND** the solution is integer, the current solution is a tour, and control goes to Step 23. Otherwise:

Step 7

The simple labelling routine of Step 4 having failed to find a violated subtour constraint, the algorithm next tries a condensation routine to identify a subtour constraint.

Step 8

If a violated constraint has been found, return to Step 2. Otherwise:

Step 9

No constraint of types (1)–(4) can be found at the current solution, but the solution is fractional. The program is considered to be in "Phase I" until the LP-optimal solution, satisfying constraints (1)–(3), over the entire set of variables, has been obtained. It has proved to be a useful heuristic procedure to defer tackling the 'integer' problem until then. Hence Step 9 checks whether the program is in Phase I. If it is not, go to Step 12. Otherwise:

Step 10

In the global check, all the variables of the problem are examined, and those which violate the duality conditions are to be added to the LP.

Step 11

If there are new variables discovered in the global check, they are added to the LP, the current solution of which, therefore, is now feasible but not optimal, and control returns to Step 2. If there are no new variables, the LP-opt solution (satisfying constraints (1) to (3) – and probably (4) – on all the variables) has been obtained, and is printed, and the program enters Phase II, at Step 12.

Step 12

Use one or two heuristic labelling schemes to try to identify a 'two-matching' constraint. No guarantee exists in this that such a constraint is found, even if there is one which is violated by the current solution.

Step 13

If Step 12 has generated a violated constraint, return to Step 2. Otherwise:

Step 14

Having failed to find any violated two-matching constraints, the algorithm will resort to the Method of Integer Forms. However, it has proved to be most satisfactory to defer this to Phase III, when all variables which could possibly enter the optimal tour solution are explicitly present in the LP. Hence Step 14 checks whether the program is in Phase III. If it is, go to Step 18. Otherwise:

Step 15

In the global check, all the variables of the problem are examined, and those which violate the duality conditions are to be added to the LP.

Step 16

As Step 11. If there are new variables, return to Step 2. Otherwise:

Step 17

In effect, the global check is carried out again, but this time all variables are added to the active LP set which could possibly form part of a tour costing less than the incumbent. The program now enters Phase III and no further recourse to the total set of variables is required.

Step 18

A single constraint of a modified version of the Method of Integer Forms[3] is added to the LP and again Step 2 is entered.

Step 19

If at Step 3 the cost of the LP solution is too great for there to be a cheaper solution than the incumbent, but the program is not yet in Phase III, go to Step 21. Otherwise:

Step 20

The incumbent – either the result of the heuristic algorithm or another and cheaper one which has replaced it during the program – is optimal.

Step 21

The program being in only Phase I or Phase II, a global check for new variables violating the duality conditions is made.

Step 22

If new variables have been found they are added to the LP formulation, and Step 2 is re-entered to re-optimize the LP. Otherwise, if there are no new variables, go to Step 20.

Step 23

The current LP solution is a genuine tour with a lower cost than the incumbent and is therefore the new incumbent. If the program is not yet in Phase III, go to Step 25. Otherwise:

Step 24

The current solution is the optimum tour.

Step 25

A global check for new variables violating the optimality conditions is made.

Step 26

If there are new variables discovered in Step 25 they are added to the LP and the program returns to Step 2. Otherwise, go to Step 24.

[2] see Figure 1

[3] see detailed explanations to follow.

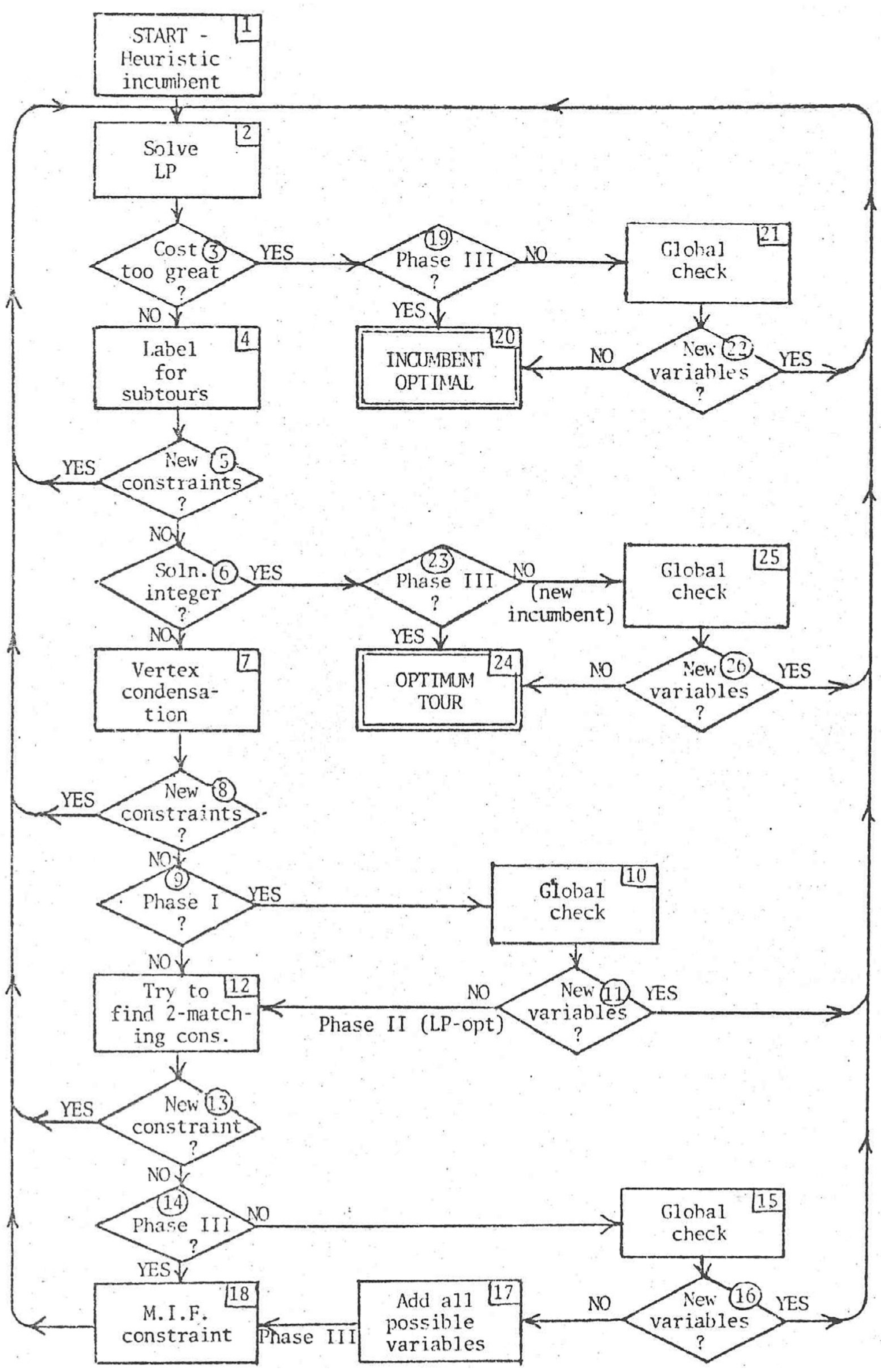


Fig. 1. Diagram 1.

Storage of coefficients in the algorithm

In the version of the algorithm used for the test problems, the raw data are the pairs of co-ordinates in the plane. A 100 'city' problem generates 4950 variables, but all of these 4950 variables are considered on during the global check stages (Steps 10, 15, 17, 21 and 25). The maximum number of 'active variables' allowed for in the program tests is 500.

The present version of the algorithm is similar to the Land-Powell [4] routines in that it uses an explicit inverse of a basis containing the currently effective constraints and original (not slack) variables. Since constraints (3) are equalities, they are always effective and therefore recorded only in the inverse and not in the original basis form.

Each variable in the current active set has recorded its two end vertices and its cost. Each one is marked as non-basic at zero, non-basic at its upper bound of one, or basic and associated with the kth row of the inverse matrix.

The coefficients of constraints (4), the subtour constraints, are not explicitly recorded, but can be deduced from the rows of a matrix of SETS. Each row of the SETS matrix represents a partitioning of the set of vertices into subsets, at least one of which has been associated with a violated constraint at some stage in the computation.

The coefficients of the two-matching contraints generated at Step 12 are recorded in two different ways – explicitly as rows of the CUTS matrix with one coefficient for each of the current active variables; and implicitly as a row of the SETS matrix, for use in the global optimality check.

Finally, the coefficients of the constraints of the Method of Integer Forms generated at Step 18 are also recorded explicitly in the CUTS matrix. Although it is also possible to record the constraints of the Method of Integer Forms in a way that enables them to be extended to the inactive set during the global check, it has proved more efficient to postpone generating these constraints until the active set has been expanded to include all variables which might possibly be in the optimum solution.

Details of the algorithm

Step 1

A '3-opt' tour is generated [5]. No experimentation has yet been carried out to see if it would be worth generating several such tours, or possibly omitting this step and only ensuring that some very simply-obtained tour was available in the first set of edges (active variables). In any complete graph TSP there are very many edges which will 'obviously' not enter an optimum solution. Here again there is room for experimentation about the choice of variables to enter the initial LP. In the computations reported here, the initial active set is simply the union of the set of edges in the heuristically generated tour with the set of edges linking each vertex to its four nearest neighbours.

Step 2

A version of the Land-Powell algorithms using exact arithmetic and an explicit (but reduced) inverse matrix is used. A feasible first basis is taken by setting all the variables which enter the tour from Step 1 to be non-basic at their upper bounds.

At every basis change the new value of the determinant is examined. If it exceeded 1,000,000 the computation would stop (as it did for some problems in earlier versions of the program). In the problems reported here the determinant never exceeded 36,192.

As the algorithm proceeds, the LP routine is re-entered, either with new, unsatisfied, constraints, or with new dual-infeasible variables. As in the Land-Powell routines, infeasibilities are eliminated sequentially and then optimality is pursued. For a TSP which has at least one feasible tour solution (as in the problems reported here), the only outcomes of the LP routine are optimality and excessive size of the determinant. The change in the criterion for selecting MIF[4] constraints seems to have avoided the latter case, though it clearly remains a possibility if the size of problem to be solved is further increased.

After re-optimization, some constraints are purged from the record. Constraints of type (4) are not purged if any of those in the whole set imposed at one time are not candidates for purging. No constraint is a candidate for purging if it is currently effective (satisfied exactly and present in the current basis). Since the various records associated with the constraints are identified by pointers, purging constraints is not a very time-consuming operation.

Step 3

In the test problems, all the cost figures are integer, hence if the optimum value of the function is strictly greater than one less than the cost of the incumbent, there cannot be a better tour solution using the current set of variables. The test could be modified to reject only LP solutions with cost greater than or equal to the cost of the incumbent.

Steps 4 and 5

The algorithm checks for violated subtour constraints (4) by labelling all vertices which are connected by edges with weights greater than one half with the same label. If there are several different labels and at least one set violates a type (4) constraint, a constraint is added to the LP for each labelled subset of the vertices for which $|S| > 2$.

The coefficients of these type (4) constraints are not recorded explicitly. Rather each vertex in a set is given a label corresponding to the associated constraint number, and the vector of these labels is stored as an added row in the SETS matrix which has space for up to 45 rows, and n_k columns. The right-hand-sides and slack vectors of the constraints are stored explicitly. A variable, x_k, associated with edge i, j has a zero coefficient in all of these constraints unless i and j have the same label, p, in which case $a_{pk} = 1$. Note that the coefficients of these constraints apply equally to variables present in the current set of active variables and to those which are considered only in the global routine. Any new variable either lies entirely within one of the sets, in which case it has a one coefficient in the constraint for that set, or i and j are in different sets and the variable has zero coefficients in this whole batch of constraints.

Step 6

If no violated constraint has been identified in Steps 4 and 5 **and** the solution is entirely integer, the current solution is a tour. By virtue of the cost test at Step 3, the tour is definitely cheaper than the incumbent tour.

Step 7

The vertices are notionally condensed into one another as the total edge weight between two 'vertices' is found to be equal to one. If at any stage a total edge weight is found between tow of these condensed 'vertices' which is greater than one, the two 'vertices' together constitute a set which violates a type (4) constraint: the current set of labelling of vertices is translated into constraint numbering, and the new set of constraints recorded as in Step 4. If the vertex condensation is completed without finding a violated type (4) constraint, it is not necessarily the case that all possible type (4) constraints are satisfied, but experience suggests that they usually are.

Step 12

If no further constraints of types (1)–(4) can be found, the solution is either a tour or is non-integer. The algorithm could at this stage enter a cutting plane routine (or a branch and bound routine, as in Miliotis' earlier paper [6]). However, a certain class of constraints which cut off non-integer points of the feasible region defined by constraints of types (1)–(4) without cutting off any integer points can sometimes be found by an 'inspection' of the solution. The simplest possible violation of such a constraint is illustrated by the configuration of the subset of three vertices in[5] Diagram 2. The three vertices i, j, k form a subset con-

[4] Method of Integer Forms, acronym introduced later in manuscript.

[5] see Fig. 2

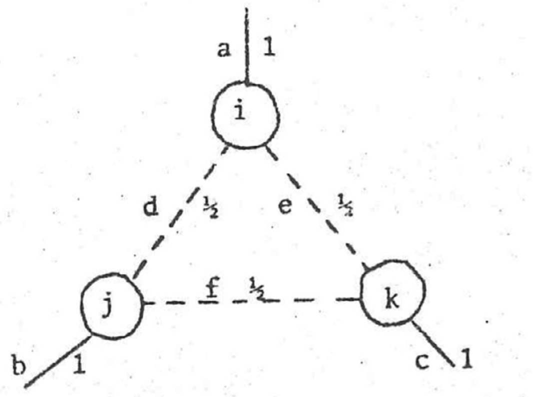


Fig. 2. Diagram 2.

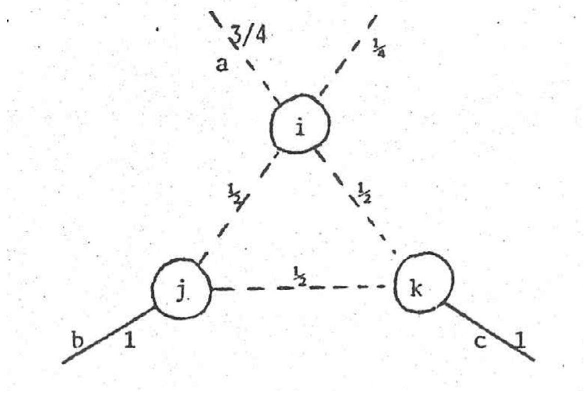


Fig. 3. Diagram 3.

nected by three edges a, b, and c, each with a weight of one, to the rest of the graph, and by 'internal' edges d, e, and f, each of weight one half. Note that the weights at each vertex total to two, and that the subtour constraint for the set i, j, k is oversatisfied ($d + e + f = 1\frac{1}{2} < 2$).

The following simple argument can be used to derive an integer cutting plane constraint. The total sum of the weights incident at vertices i, j, k must be six (two at each vertex). The **maximum** contribution which can be made by edges a, b and c is three (since they each have an upper bound of one and each contributes only one to the total of six). Therefore **all other** edges incident to i, j, k must contribute at least three units of weight. However, an edge weight of one of these other edges **may** contribute two units to the weights incident at i, j, k (e.g., a weight of one on edge d would contribute one unit to the incidence at i and one to the incidence at j). Hence:

2(sum of weights on all edges incident to i, j, k **except** a, b, c) ≥ 3 .

Dividing the constraint by two:

(sum of weights on all edges incident to i, j, k **except** a, b, c) $\geq 1\frac{1}{2}$.

But the weights on all edges must be integer in a TSP solution, therefore:

(sum of weights on all edges incident to i, j, k **except** a, b, c) ≥ 2 .

Since the only edges specified in this constraint which are non-zero in the configuration of Diagram 2 are d, e, and f, which have a total weight of $1\frac{1}{2}$, it is evident that this is a cutting constraint at this stage of the computation. The constraint is exactly equivalent to a simple case of a 'two-matching' constraint [1].

The extension of this argument to larger sets involves finding subsets of vertices which are connected to the rest of the graph by an odd number of edges with weights of one. Occasionally, also, one may find such a constraint when a single edge with a weight greater than one half, but less than one, is treated as one of the unit edges, as in[6] Diagram 3. The constraint is still:

(sum of weights on all edges incident to i, j, k **except** a, b, c) ≥ 2 .

This time, the sum for the constraints at the current solution is $1\frac{3}{4}$ instead of $1\frac{1}{2}$ as in Diagram 2.

Unfortunately, no guaranteed method of finding such violated constraints was found, and in any case it is known [3] that two-matching constraints are not sufficient to force an integer solution. A great deal of time and ingenuity was expended, but in the end the time spent in the algorithm on trying different labelling schemes proved to be less fruitful than abandoning the search after only one trial heuristic labelling.

The coefficients of these constraints are recorded in two ways: firstly as a simple explicit row of n_k zeros and ones as a row of the CUTS matrix; secondly as a set of vertices in the same format as type (4) constraints, for later use in the global routine. In this case, the coefficient of any variable is one in the constraint if it has **one or both** of its vertices belonging to the set **unless** it is one of the **special** edges (like a, b and c in the example) which has a zero coefficient. This means that any variable which was not present in the active set when the constraint was generated cannot be one of the special edges, and therefore can have its coefficient determined solely by whether or not it has at least one vertex within the set. It also means, however, that care must be taken never to delete a special variable from the active set, or to introduce it again in the global routine. In order to prevent either of these two eventualities, the variable is labelled as a special one.

[6] see Fig. 3

Step 14

The algorithm has run out of new constraints and the solution is still not integer. If Phase III has been reached, all possible variables are explicitly present in the LP and MIF constraints will be tried next. Otherwise a search will be made through the entire set of variables in the global routine to discover if any variables from the inactive set violate the dual conditions.

Steps 10, 15, 21 and 25

In these steps, the check for global optimality is carried out. The two-matching constraints have been recorded in two ways: directly in the CUTS matrix, and indirectly in vertex sets like the tour-barring constraints. Throughout the algorithm the required non-zero elements of a column of the A matrix[7] are obtained in a subroutine. Except for the variables labelled as special in Step 12, the two ways of computing the coefficients for any currently active variable must be equivalent. Since the purging operation, although not time-consuming, is messy, caution requires that before using the second form of the constraints in the global routine a check should be run to ensure that for every non-special variable in the current set of n_k variables the two columns are the same in the rows of the effective constraints. This check has therefore been carried out every time the global routine is entered.

In the test problems, the only record of the costs of the variables not in the current set is their co-ordinates in the plane. Thus variables will be generated and checked for optimality even if they are already present in the current set. For ordinary basic variables, and for variables non-basic at zero, this will merely result in redundant calculations. But for two sets of variables in the current set there is a risk that the computation of dual feasibility in the global check will discover a spurious dual infeasibility: the two sets are variables that are non-basic at their upper bounds, and the 'special' variables of the two-matching constraints.

Since there is no way of identifying the current variable numbers from their vertex identities, some way of stopping these spurious infeasibilities is required. It would be too clumsy to go right through the list

[7] notation: see appendix

of variables to discover for each new variable whether it was one already present. The method used is to organize as many as possible of the special variables and variables non-basic at their upper bounds into ordered trees, so that every new variable can be immediately skipped over if either of its vertices is the predecessor of its other vertex in the tree. Those few variables which will not fit into a tree are recorded in a separate list, and an identifier of how many times a vertex appears in this list ensures that the list is inspected only when necessary. The number of these surplus variables never exceeded 50 in the problems reported here.

For each i and j, $i < j$, which is not skipped over by the preceding consideration, all non-zero elements of the column of coefficients are computed, and the dual feasibility check carried out. After filling up any surplus space for variables (500 in these computations), further new variables take the place of (non-special) variables which are non-basic at zero in the current calculations, and hence are eliminated.

Step 17

If the global check has been entered from Step 14 and no new variables have been found, the algorithm is about to introduce MIF constraints. To avoid keeping an 'implicit' record of these constraints, it has proved to be more efficient to bring into the active set at this stage every variable which has 'reduced cost' less than the difference between the cost of the current LP solution and that of the incumbent. No variable with a greater violation of dual feasibility can enter the optimum solution. (It would be possible to similarly eliminate from further change also some of the variables which are non-basic at their upper bounds, or at least to label them to be skipped over by the LP computation, but this has not been done.)

In effect, Step 17 is a repetition of the immediately previous global check with a more generous limit for the introduction of new variables, though it also requires a larger class of 'special' variables to prevent the re-introduction of existing variables.

Step 18

The solution being non-integer, and no new variables or constraints being discoverable, a version of Gomory's Method of Integer Forms (MIF) is applied. As in the Land-Powell algorithms, the coefficients of the MIF constraints are obtained and recorded in the `CUTS` matrix in terms of the original variables of the problem, not only in the updated form in the inverse.

An earlier version of the algorithm sometimes failed after several MIF constraints were applied because the value of the determinant exceeded the limit set by the word size of the computer. This is a typical result of using an exact arithmetic version of a cutting plane algorithm for general ILP problems. However, an alteration in the criterion used to select amongst possible MIF constraints has proved to be remarkably effective in the TSP case. (The effect of using the new criterion remains to be tested on other types of ILP problems.)

In general ILP terms, if an MIF constraint is generated from a row of the updated simplex tableau, a whole group of different constraints can be generated by multiplying the row by integers from 1 to k, where $k \leq D - 1$ (D is the determinant of the basis). In TSP, where there are far fewer fractional values than $D - 1$, k is a very much more modest number than $D - 1$. A solution with determinant, say, 192, may have edge weights (values of basic variables) of only 0,32,64,96,128,160, and 192, i.e., $0, \frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}$, and 1. (It is, of course, well-known that the solution to a TSP with only the constraints (1)–(3) may have a determinant with a large power of two, whereas the values of the variables will be only $0, \frac{1}{2}$, or 1.)

This algorithm takes as a *base row*[8] one which has a right-hand-side (RHS), t, equal to the HCF[9] of all the (numerators of the) variables and the determinant. If there is no such row, one is generated by adding together two rows which have RHS's with HCF of t and multiplying by the appropriate integer to achieve the required row. Then there are $\frac{D}{t} - 1$ possible different constraints which can be generated from the base row. The algorithm will proceed to generate constraints until it finds one for which the largest absolute value of the coefficients (expressed in the original variables) is one; or failing that, to find the constraint from the set with smallest maximum absolute value of a coefficient. In the problems reported here, the largest maximum absolute value found was 24.

It is perfectly possible that when the constraint has maximum value of one, it could be a constraint of the same type as those discovered in Step 12, and hence could be recorded in the same way. However, no attempt has been made so far to identify the sets of vertices associated with these MIF constraints.

After selecting the constraint with the smallest coefficient, in fact the constraint is 'tightened' by taking account of the bound on the function provided by the incumbent (see Appendix). This means that the cuts are actually 'deeper' than those provided by facets of the travelling salesman polytope. The limiting polytope is the convex hull of the tours with costs strictly less than that of the incumbent. If the incumbent is in fact the optimum, that polytope may be null.

Computational results

To test the program, ten 100-'city' problems have been generated from 100 points with two co-ordinates selected at random in the range 0.0 to 100.0. The distances between pairs of points are rounded to the nearest integer. Thus the cost matrix consists of integer elements in the range 0 to 141. In fact only those distances corresponding to variables in the current active set are stored: the others are computed each time the global routine is entered.

It has been observed (e.g. [7]) that problems in which the cost matrix elements are random tend to be much easier to solve than ones in which the triangular inequalities are satisfied. To test this observation using these programs, three further symmetric problems (numbers 11, 12 and 13 in the following table) were tried. In order to make the distribution of cost elements similar to those in the ten Euclidean problems, each cost element was obtained by generating 4 random numbers in the range 0.0 to 100.0, p_1, p_2, p_3, p_4, and taking

$$c_{ij} = \sqrt{(p_1 - p_2)^2 + (p_3 - p_4)^2}\,.$$

These numbers were re-computed at each global check by ensuring the same starting random seed each time.

The results of the computations are summarized in the table.[10] The problems were generated using the `CDC FORTRAN` facility:

```
CALL RANSET(T)
 XI = RANF(0.0)
          etc.
```

The random number seed, the value for T, for each problem is recorded with the problem number.

The value of the 3-opt tour is shown, and, out of interest really generated by problems 11, 12 and 13, the value of the very first LP solved, i.e., the solution to the constraints of types (1)–(3) only (the 'assignment constraints') on the first set of active variables (column headed 'limited assignment').

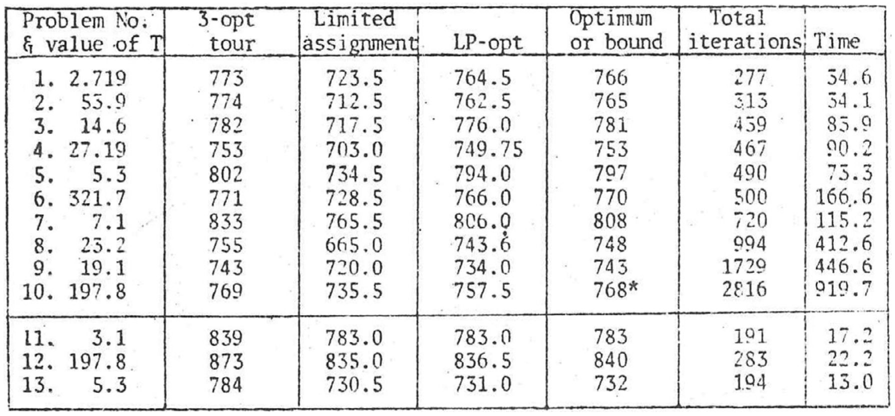
The column 'LP-opt' shows the value of the function when all constraints of types (1)–(4) are presumed to be satisfied on all the variables – just as Phase II is entered. (There may be, however, some violated tour-barring constraints which have not been discovered.)

The column 'optimum or bound' shows the value of the optimum solution in every case except Problem 10 (which is discussed below).

[8] emphasis by editor

[9] highest common factor = greatest common divisor (GCD)

[10] see Fig. 4



Problem No. & value of T	3-opt tour	Limited assignment	LP-opt	Optimum or bound	Total iterations	Time
1. 2.719	773	723.5	764.5	766	277	34.6
2. 53.9	774	712.5	762.5	765	313	34.1
3. 14.6	782	717.5	776.0	781	439	85.9
4. 27.19	753	703.0	749.75	753	467	90.2
5. 5.3	802	734.5	794.0	797	490	73.3
6. 321.7	771	728.5	766.0	770	500	166.6
7. 7.1	833	765.5	806.0	808	720	115.2
8. 23.2	755	665.0	743.6	748	994	412.6
9. 19.1	743	720.0	734.0	743	1729	446.6
10. 197.8	769	735.5	757.5	768*	2816	919.7
11. 3.1	839	783.0	783.0	783	191	17.2
12. 197.8	873	835.0	836.5	840	283	22.2
13. 5.3	784	730.5	731.0	732	194	13.0

Fig. 4. Table of results

'Total iterations' are simply the total number of simplex iterations, and 'Time' is the number of seconds on the London University CDC 7600 (FTN compiler) from the first heuristic solution. The ten Euclidean problems are numbered in order from the easiest to the most difficult (in terms of the number of simplex iterations).

It is very noticeable that the range in 'difficulty' is very great, with Problem 10 taking more than ten times as many iterations as Problem 1.

Problem 10 is the only one which failed to reach an unambiguous answer. The 3-opt tour costs 769. After 1096 iterations (and 398.2 seconds) it was established that the optimal function value was definitely greater than 767. After 625.5 seconds the optimal LP solution was 768 exactly, and remained at that value (with solution values given by halves, quarters, occasionally twelfths) as constraints were added and purged, but never either found an integer solution at 768 nor proved the cost to be greater than 768 in the remaining time before the computation was abandoned. Probably a modest amount of 'branching and bounding' would quickly establish whether there is or is not a solution at 768 for this problem. Alternatively, a less crude 'integerizing' of the individual cost elements might have arrived at a less ambiguous answer.

The application of the algorithm to the three random cost problems, numbers 11, 12 and 13, produced rather startling statistics. For one of the problems (No. 11) the very first 'assignment' solution, constrained only by constraints of types (1)–(3), and using only the first set of active variables, turned out actually to satisfy the conditions for a tour. In the other two cases, the first solution value, the LP solution value, and the optimal value were all very close. Each of the three was very much easier to solve than any of the Euclidean problems – despite the fact that the initial 3-opt tour was uniformly worse (percent of optimum) than for the ten Euclidean problems.

Conclusions

An appropriately designed cutting plane algorithm can be applied with considerable success to the symmetric travelling salesman problem. If only a good bound is required (within 0.6% of the optimum for all of these problems) even the LP solution, which is very readily obtained, may be adequate. Unfortunately, a record was not made of the almost as readily available bound achievable by adding some two-matching constraints. Clearly, they go some way to closing the gap between the LP-opt and the optimal tour solutions.

A secondary conclusion must be a warning against using random cost matrices when assessing the value of a TSP algorithm.

Declaration of Competing Interest

No conflict of interest declared.

Appendix A. Use of a bound in a cutting plane algorithm

A 'good' solution to an ILP problem, obtained either by a heuristic algorithm, or during the course of an optimizing algorithm, has always been of use in a branch and bound algorithm. This note describes how it can be also of use in a cutting plane algorithm.

Let us consider the problem

$$\text{Max. } cx\,,\ \text{ s.t. } Ax \leq b\,,\ x \geq 0\,,\ \text{ and } x \text{ integer.}$$

Gomory's Method of Integer Forms [2] as implemented in the Land and Powell programs [4] solves a relaxed LP problem, and then adds additional constraints of the form:

$$px \leq \beta\,,$$

such that p is an all integer vector,[11] and β is an integer scalar. The constraint is so chosen that $z = px$ is maximized at a value $\beta + f_0$ (where $0 < f_0 < 1$) at the LP optimum point. Thus $px \leq \beta$ is a constraint which cuts off the LP optimum point, but not any feasible integer point.

Customarily, a branch and bound algorithm excludes the current LP optimum point by the addition of two alternative constraints,

$$px \leq \beta \quad \text{and} \quad px \geq \beta + 1\,.$$

Thus after 'branching' there are two alternative candidate LP problems to be fathomed. Usually, but not essentially (see Brocklehurst [ ])[12], these constraints are of the form

$$e^j x \leq \beta \quad \text{and} \quad e^j x \geq \beta + 1\,,$$

where e^j is the jth unit vector.[11]

Thus a cutting plane can be regarded as a special case of branching in which the LP defined by the addition of the constraint

$$px \geq \beta + 1$$

is known to be infeasible.

The case considered here is that in which the LP with the constraint

$$px \geq \beta + 1$$

can be excluded, not by virtue of its infeasibility, but because the optimum value of the LP so defined[13] is necessarily less than a known integer solution: it can be excluded by comparison with a bounding value of the objective function.

[11] *viz.* row vector

[12] empty bracket in original, presumably [8] is meant

[13] with objective cx and adding the constraint $px \geq \beta + 1$

Consider a polyhedral feasible region expressed entirely in less-than-or-equal constraints, with the non-negativity constraints also explicitly expressed as less-than-or-equal constraints. Note that this formulation is always valid, since a greater-than-or-equal constraint can be converted to a less-than-or-equal constraint by multiplication by -1, and an equality constraint can be replaced by two inequalities. Assume also that all coefficients in A and b are integers.

$$\left.\begin{array}{rcr} Ax & \leq & b \\ -Ix & \leq & 0 \end{array}\right\} \text{ where } A \text{ is of order } m \times n\,.$$

A basis in this formulation corresponds to a selection of n independent inequalities to be solved as equalities:[14]

$$\left.\begin{bmatrix} R & U \\ 0 & -\hat{I} \end{bmatrix} \begin{bmatrix} x^R \\ x^U \end{bmatrix} = \begin{bmatrix} b^R \\ 0 \end{bmatrix}\right\} \quad n \text{ equalities.}$$

where, after possible rearrangement, x^R and x^U together constitute x and $\hat{I}$ represents an appropriate part of the $n \times n$ identity matrix I. The basis yields a feasible solution, $\bar{x}$, if the remaining m inequalities are satisfied:

$$\left.\begin{bmatrix} S & V \\ -\tilde{I} & 0 \end{bmatrix} \begin{bmatrix} x^R \\ x^U \end{bmatrix} \leq \begin{bmatrix} b^S \\ 0 \end{bmatrix}\right\} \quad m \text{ inequalities.}$$

A linear function, px, is maximal at $\bar{x}$ if p can be expressed as a non-negative weighted multiple, $\bar{y}$, of the rows of

$$P = \begin{bmatrix} R & U \\ 0 & -\hat{I} \end{bmatrix}.$$

i.e., if $\bar{y}P = p$, $\bar{y} \geq 0$.

Gomory's Method of Integer Forms can be regarded as a method of finding a vector p of integer coefficients which reaches a non-integer maximum, $p\bar{x} = \beta + f_0$, at $\bar{x}$, and hence enables us to cut off the fractional part, f_0, to yield a new constraint, $px \leq \beta$ (see [4, p.25-28]).

The vector $\bar{y}$ of the non-negative fractional parts of any row (or multiple or sum of rows) of P^{-1}, so long as the associated element of $\bar{x}$ is non-integer, is such a vector of weights.

A negative element in $\bar{y}$ indicates that the function px is not maximal at $\bar{x}$, but that $p\bar{x}$ could be increased by 'moving along the edge of the feasible region' defined by relaxing the constraint corresponding to the negative element. Thus if there are negative elements in $\bar{y}$ we have to consider not only the constraint $px \leq \beta$, but also the other branch, $px \geq \beta + 1$. However, as in a standard branch and bound implementation, we have an immediate 'penalty' calculation which yields a lower bound on the cost of satisfying that alternative constraint. If that bound exceeds the difference between the current LP solution and a known feasible integer solution we can ignore the branch $px \geq \beta + 1$, and validly impose the constraint $px \leq \beta$.

In practice, instead of taking simply the fractional parts, $\bar{y}_j = f_j$, from a row, one considers for each element the effect of subtracting one (or more) from the fractional part. So long as

$$\bar{y}_j \geq (f_0 - 1)\Pi_j/\delta$$

(where Π_j is the jth element of the maximized function row, and δ is the difference[15] between the current function value and the best known integer solution) the alternative branch can be ignored. Only integer values may be subtracted from f_j to ensure p integer.

References

[1] J. Edmonds 'Maximum matching and a polyhedron with 0,1 vertices' Journal of Research of the National Bureau of Standards Sect. B69 (1965), 125–130.

[2] R. Gomory 'An algorithm for integer solutions to linear programs' in R.L. Graves and P. Wolfe, eds., 'Recent Advances in Mathematical Programming' (McGraw-Hill, New York, 1963) 269–302.

[3] M. Grötschel and M.W. Padberg 'On the symmetric travelling salesman problem I: inequalities' Mathematical Programming 16 (1979), 265–280.

[4] A. Land and S. Powell 'Fortran Codes for Mathematical Programming' Wiley 1973.

[5] S. Lin 'Computer solution of the travelling salesman problem' Bell System Technical Journal 44 (1965), 2245–2269.

[6] P. Miliotis 'Integer programming approaches to the travelling salesman problem' Mathematical Programming 10 (1976), 367–378.

[7] P. Miliotis 'Using cutting planes to solve the symmetric travelling salesman problem' Mathematical Programming 15 (1978), 177–188.

[8] E. R. Brocklehurst 'Generalized branch and bound: a method for integer linear programs' in K.B. Haley, ed., 'Operational Research '75. Proceedings of the 7th IFORS International Conference on Operational Research, Tokyo, Kyoto 1975 (North-Holland, Amsterdam, 1976), 141–162.

[9] D. Shier 'Ailsa H. Land, EURO Gold Medallist (1927–2021)' European Journal of Operational Research 296 (2022), 1–2.

[14] this part slightly re-edited to improve readability

[15] symbol diff is used *en lieu* δ in original text