A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation

Ton VOLGENANT and Roy JONKER
Instituut voor Acturiaat en Econometrie, University of Amsterdam, 1011 NH Amsterdam, Netherlands

Received October 1980
Revised January 1981

In 1970 Held and Karp introduced the Lagrangean approach to the symmetric traveling salesman problem. We use this 1-tree relaxation in a new branch and bound algorithm. It differs from other algorithms not only in the branching scheme, but also in the ascent method to calculate the 1-tree bounds. Furthermore we determine heuristic solutions throughout the computations to provide upperbounds. We present computational results for both a depth-first and a breadth-first version of our algorithm. On the average our results on a number of Euclidean problems from the literature are obtained in about 60% less 1-trees than the best known algorithm based on the 1-tree relaxation. For random table problems (up to 100 cities) the average results are also satisfactory.

1. Introduction

There are many algorithms to solve the symmetric traveling salesman problem (TSP). Our main purpose was to construct a simple and fast algorithm based on existing algorithms, but extended with several new ideas. For us the most important algorithm was the algorithm of Held and Karp [8], the first one based on the minimal spanning 1-tree as a lower bound for the TSP. In the following sections we will describe our algorithm, that gives one of possibly more optimal solutions for a TSP. We introduce a straightforward branching scheme; our ascent method for the lower bounds is characterized by only a small number of parameters. We developed a heuristic that is a combination of an algorithm of Christofides [2] and one of Lin [12], but both simplified. The most important measure for the efficiency is for us the number of minimal spanning 1-trees that have to be determined, as most of the computational effort in the 1-tree type TSP algorithms is spent constructing these trees. The computational results are very satisfying for Euclidean problems: on the average we have to solve less than half the number of 1-trees that have to be computed by the best algorithm to date. We only slightly improve the best results for random table problems.

We thank Mr. Gerard Kindervater for his valuable help at the computational side of this article.

2. The Lagrangean approach

In this section we briefly describe the basic ideas of the Lagrangean approach to determine TSP lower bounds from minimal spanning 1-trees.

Let G be a complete undirected graph with node set $N=\{1,2,\dots,n\}$. Each edge (i,j) with $i,j\in N$ of G has an associated length c_{ij}; the matrix $C=(c_{ij})$ is the distance matrix.

A minimal (spanning) 1-tree of G consists of a spanning tree of minimal total length on the node set $N\setminus\{1\}$ combined with the shortest two edges incident to node 1. The choice of node 1 as a special node is arbitrary. The number of edges of a 1-tree (say with sequence number m) incident to a node j is called the degree d_j^m of j.

Obviously a distance transformation $c_{ij}:=c_{ij}+\pi_i+\pi_j$ has no influence on the relative order of the TSP solutions. However after the transformations we will generally find a different minimal 1-tree.

The TSP can be formulated as: determine a minimal 1-tree under the restrictions that the degree of every node equals two. The π_i $(i\in N)$ can then be interpreted as multipliers associated with the degree constraints.

Using standard Lagrangean theory, see for instance [4], it is clear that for a set of multipliers π_i $(i\in N)$ a lower bound for the TSP is given by

$$w(\pi)=z(\mathrm{T})+\sum_{i\in N}\pi_i(d_i-2),$$

with $z(T)$ the length of the corresponding minimal 1-tree. The best bound of this type is $w^* = \max w(\pi)$.

Held and Karp [7] and Christofides [1] independently suggested ascent methods for finding a good set of multipliers to approximate w^*. In both papers a branch-and-bound approach is used to close the so-called duality gap that may exist between w^* and the optimal TSP value. Held and Karp [8], later on Helbig Hansen and Krarup [5] and almost at the same time Smith and Thompson [14] introduced valuable improvements on the algorithm as a whole as well as on details.

3. Ascent methods

Held and Karp's ascent method [8] was refined by Held, Wolfe and Crowder [9]. A variant of their method is used in the TSP-algorithm of Smith and Thompson [14]. A single iteration of this ascent method can be described as follows. Given a set of multipliers $\{\pi_i^m, i \in N\}$ with m a sequence number ($m = 1,2,\ldots$), compute a minimal 1-tree T with respect to the transformed edge lengths $c_{ij} + \pi_i^m + \pi_j^m$. Let L be the corresponding lower bound and let U be an upper bound on the optimal tour length. If T is a tour, the ascent is terminated since L is the optimal TSP value, otherwise let p be a positive scalar not greater than 2. The new multipliers $\{\pi_i^{m+1}, i \in \mathrm{N}\}$ are computed according to

$$\pi_i^{m+1} = \pi_i^m + t^m(d_i^m - 2) \quad (m = 1,2,\ldots)$$

with

$$t^m = \frac{p(U-L)}{\sum_{j\in N} (d_j^m - 2)^2} \quad (m = 1,2,\ldots)$$

and

$$\pi_i^1 = 0 \quad (i \in N).$$

Held, Wolfe and Crowder proved this ascent to converge if U is the length of an optimal tour and p tends to zero.

As the convergence rate is low and moreover the value of U is generally unknown, we have chosen for a different method for the computation of π_i^{m+1}. We use the formula

$$\pi_i^{m+1} = \pi_i^m + 0.6t^m(d_i^m - 2) + 0.4t^m(d_i^{m-1} - 2) \quad (i \in N, m = 1,2,\ldots)$$

for points with $d_i^m \neq 2$, and with $\pi_i^1 = 0$ ($i \in N$).

Using the update formula of Held et al we observed frequently oscillation of the degree of a point (between one and greater than two). To suppress this oscillation we introduced successfully the damping term with $(\mathrm{d}_i^{m-1} - 2)$ in our formula. We obtained not only higher lower bounds but also more, different 'well-shaped' 1-trees, which is very important for our upper bound heuristic as explained in Section 5.

In the update formula t^m is a positive scalar decreasing during the ascent according to a series with constant second order difference:

$$t^{m+1} - 2t^m + t^{m-1} = \text{constant}.$$

Intuitively this approach is clear: we start with some large stepsizes to obtain roughly correct multipliers and then soon shift gradually to a large number of smaller stepsizes that take care of the minor corrections. A geometric series would probably have worked as well.

Furthermore the ascent has a fixed length M, in contrast with the rather cumbersome stopping rules used in other TSP-algorithms.

With Helbig Hansen and Krarup [5] we distinguish between the use of the ascent method on the general problem (the initial ascent) and its use on subproblems generated subsequently in the branching process (general ascents).

After some experimenting we fixed the maximum number of steps on $[n^2/50 + \frac{1}{2}] + n + 15$ for the initial ascent. Fixing this maximum number of steps is based on the idea that the profit of steps after this maximum number is so small that it is better to branch the problem first.

For the initial ascent we chose a fixed number that is a quadratic function of n because of the empirical observation that a linear function is either insufficient for large n or inefficient for small n. Intuitively it is clear that if the initial ascent finds a good set of multipliers a general ascent should require fewer ascent iterations than the initial ascent to find a good set of multipliers for the subproblem under consideration.

After experiments on a set of various problem types and sizes the linear function $[n/4] + 5$ proved to be a sufficient number of iterations.

The details of the initial and the general ascent are given by the following points:

(a) at the first multiplicator change:

$$d_i^0 = d_i^1, \quad i \in N;$$

(b) in the initial ascent

$$t^1 = 0.01\, l^*$$

with l^* the length of the best known minimal 1-tree during this ascent (the value of t^1 is updated each time a better value for l^* is found); introducing l^* into this formula performs a scaling function.

(c) in the general ascent

$$t^1 = 0.5 \sum_{i \in N} |\pi_i^*|/n$$

with π^* the set of best multipliers in the initial ascent; intuitively it is correct to chose a stepsize that is a function of the now known range of the multipliers. On our set of diverse test problems the constant 0.5 performed best on the average.

(d) in steps where the current lower and upper bound are 'close' together (in our algorithm if $t^m > u - l$) the purpose of a general ascent changes; it shifts from finding a high value of l to finding a value for l with $l > u - 1$. For that reason we omit steps with $t^m > u - l$;

(e) the second order difference has been chosen such that

$$t^1 - t^2 = 3(t^{M-1} - t^M), \quad \text{with } t^M = 0;$$

again the constant has been determined empirically.

4. The branching scheme

A good branching scheme leads to a partitioning of the current set of feasible solutions into disjoint subsets. It is desirable that the branching improves the lower bounds for each of the newly created subsets by excluding the current best solution. In traveling salesman problems branching takes place by requiring and/or forbidding certain edges, so that each subset is characterized by a set R of required edges and a set F of forbidden edges.

Throughout the computations two simple rules are used to extend these sets of required and forbidden edges:

(a) if two edges incident to a point are required, all other edges incident to this point can be forbidden;

(b) if there are no more than two not forbidden edges incident to a point both edges are required. In fact this comes to requiring the considered point to have degree two.

Our branching scheme starts by determining a point, say p, with degree greater than two and, for simplicity, on the subtour of the current best 1-tree. As a consequence of rule (a) there are always at least two non-required edges in this 1-tree incident to point p, say $e1$ and $e2$.

Let $S(R, F)$ be the current set of feasible solutions with R the set of required and F the set of forbidden edges.

Now $\{S1, S2, S3\}$ is the branching partitioning of $S(R, F)$ with

$$S1 = S1(R \cup \{e1, e2\}, F),$$

$$S2 = S2(R \cup \{e1\}, F \cup \{e2\}),$$

$$S3 = S3(R, F \cup \{e1\}).$$

If point p is already incident to a required edge we know subset $S1$ to be infeasible: this leads to a branching into two subsets.

From our computational experience we destillated three heuristic rules for the choice of p, $e1$ and $e2$:

(1) if possible we take p to be incident to a required edge as we prefer a branching into two subsets;

(2) if there still is a choice for p we take the point with the smallest number of feasible edges incident to it; intuitively it is clear that this rule gives a smaller chance of fixing in a wrong edge;

(3) we take $e1$ not to be part of the subtour; in general and especially in our depth-first version it turns out that this leads faster to encountering better solutions.

5. The upper bound heuristic

We determine upper bounds on the optimal solution using a combination of two well-known heuristics for the TSP, both simplified. Christofides [2] developed a heuristic based on combining a minimum 1-tree and a perfect matching on its odd degreed points. Through this combination of a 1-tree and a matching a tour is determined which may contain points more than once. A feasible tour is obtained by deleting from this sequence every point that has been encountered before.

This type of heuristic is very well suited within the framework of our algorithm: it can be used on every 1-tree available during the solution procedure. However, we confine its use to 1-trees con-

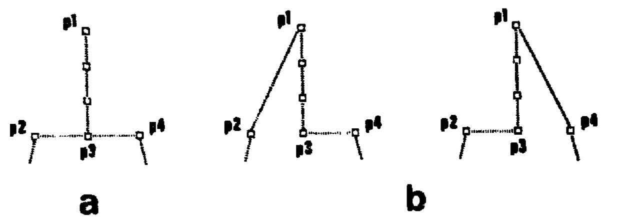

Fig. 1. The modification of a part of a 1-tree into a part of a tour; (a) part of a 1-tree; (b) two ways of making a part of a tour.

sisting of a cycle with only branches that have points with degree not greater than two. This enables us to solve Christofides' perfect matching heuristically by a simple rule: for every branch we match the point with degree 1 and its point on the cycle. For each branch—with one point with degree 1, say $p1$, and one point on the cycle, say $p3$ —there are two points, say $p2$ and $p4$, on the cycle connected to the point $p3$ (Fig. 1(a)).

Now we chose the cheapest way out of two to construct a part of the tour, namely the sequence

$$p2\text{-}p1\text{-}\cdots\text{-}p3\text{-}p4 \text{ or } p2\text{-}p3\text{-}\cdots\text{-}p1\text{-}p4 \text{ (Fig. 1)}.$$

For the current length l of the heuristic solution under construction this comes to

$$l := l + \min\{c_{p1p2} - c_{p2p3}, c_{p1p4} - c_{p3p4}\}.$$

If $l \geq u$ (with u the current upper bound) the search for a better upper bound can be terminated except

(1) during the initial ascent, where we store the best three solutions during this ascent; these three tours are subjected to the second part of our heuristic;

(2) if $p2$ or $p4$ has degree greater than two: in this case the change of l in the next step of the heuristic procedure may turn out to be negative.

The best three heuristic solutions in the initial ascent and all solutions better than the current best in the general ascents are subjected to an improvement algorithm that is a modification of the Lin algorithm [12]. The search for three edges to move out of the current solution is restricted to cases with at least two edges incident to each other. This is equivalent to improvement by insertion combined with 2-optimality. The algorithm finishes if no better insertion is found for all n cities.

The heuristic as a whole is modest in computer time consumption. For all solved problems the time spent on it is on the average about 10% of the total solution time. An indication for the quality of the combined heuristic for Euclidean problems is its performance during the initial ascent: on the average only 0.15% above the optimal value for the 18 problems used by Smith and Thompson [14]. Their heuristic algorithm (the Karg–Thompson–Raymond algorithm) is on the average 0.51% above the optimal value for the same problems. Like most heuristic TSP algorithms this one is not suited for random table problems. For this type of problems (see section 6) the algorithm gave a solution within 1% of the optimal value in 1 problem with $n = 60$, 3 problems with $n = 80$ and 3 problems with $n = 100$ out of 15 problems for each value of n.

6. Computational results

In the Tables 1 and 2 we will compare our computational results with those of Smith and Thompson [14]. With them we consider the number of computed 1-trees a better measure to evaluate algorithms based on the 1-tree relaxation than total run time. This first measure avoids the difficulties of comparing different computers and codes. Helbig Hansen and Krarup [5] did not record statistics such as number of 1-trees, but their results do not differ very much from those of Smith and Thompson [14] and therefore have been left out.

We solved the same Euclidean problems as Smith and Thompson [14] did; in the first place their problems R 481 to R 485 (48 cities) and R 600 to R 609 (60 cities) and secondly the classical problems DF 42 and KT 57, that appear in [10], and HK 48 [6].

We did not solve graph problems. In our opinion an algorithm should be adapted especially for this type of problems.

The random table problems are generated in the same way as in [5], i.e. the distance matrix entries were drawn from a discrete uniform distribution over the interval [1,1000]. For $n = 60$ 5 problems, for $n = 80$ 2 problems and for $n = 100$ 3 problems were solved by the initial ascent, for all three values of n out of the generated 15 problems.

The initial ascent of Smith and Thompson solved relatively more problems: 6 out of 15 problems for $n = 60$, 5 out of 15 for $n = 70$ and 2 out of 5 for $n = 80$ (we consider this sample too small). An explanation for this behaviour may be the

Table !
Computational results for Euclidean problems

problem		relative gap		total number of 1-trees			total cpu-time		
		lower	upper	ST	df	bf	ST	df	bf
DF	42	0.30	0	182	147	147	6	1.3	1.4
HK	48	0.17	0	234	117	117	14	1.5	1.6
R	481	1.88	1.04	9588	3792	2481	396	31.9	23.6
	2	0.20	0	304	150	150	16	1.8	2.0
	3	0.06	0	197	121	121	12	1.4	1.5
	4	0.67	0.29	529	255	228	25	2.5	2.4
	5	0.17	0	262	137	137	15	1.5	1.6
KT	57	0.37	0.23	1439	408	409	88	5.0	5.5
R	600	0.08	0	1286	172	172	94	2.9	2.9
	1	0.99	0.04	4064	1129	1123	257	14.2	15.7
	2	0	0	128	130	130	16	2.0	2.0
	3	1.01	0.49	7106	2600	2397	423	33.0	33.6
	4	1.15	0.24	675	286	299	50	4.2	4.6
	5	0.87	0.13	10570	3849	3081	654	46.1	42.7
	6	0.03	0.09	312	193	200	26	3.1	3.5
	7	0.47	0.20	3715	653	817	232	8.2	10.4
	8	1.13	0.02	2319	1074	1063	147	13.0	14.1
	9	0.79	0	314	250	250	27	3.8	3.9

The columns:
relative gap: the difference between the bounds after the initial ascent and the optimal value as a percentage of this value;
ST: results from Smith and Thompson [14];
df: results of the depth-first version;
bf: results of the breadth-first version.
The total cpu-time (in sec) includes the time for the computation of upper bounds.

Table 2
Computational results for random table problems

number of cities		total number of 1-trees			total cpu-time		
		ST	df	bf	ST	df	bf
60	average	572	352	265	34	4.7	4.1
	average*	906	461	331	54	6.1	5.1
	st. dev.	796	277	165	47	3.4	2.4
	maximum	3211	1052	728	191	13.2	10.5
80	average	764	702	640	83	15.5	15.6
	average*	1195	779	707	130	17.2	17.2
	st. dev.	908	526	590	99	10.9	13.3
	maximum	2530	2102	2635	275	45.0	60.3
100	average	–	1664	–	–	53.2	–
	average*	–	2003	–	–	63.5	–
	st. dev.	–	1828	–	–	53.6	–
	maximum	–	7510	–	–	222.2	–

The columns:
ST: results from Smith and Thompson [14];
df: results of the depth-first version;
bf: results of the breadth-first version.
The total cpu-time (in sec) includes the time for computation of upper bounds for the df- and bf-column.
The *-averages are calculated without the problems solved during the initial ascent.

relatively small values of the step sizes early in their initial ascent.

As already mentioned our heuristic did usually not provide a good upper bound for random table problems. For this reason we used as upper bound 1.01 times the value of the lower bound at the end of the initial ascent, a well known approach, also used by Smith and Thompson. As a consequence the algorithm has to be run again with a higher upper bound if no tour is found. (This case did occur only once, in an 80-city problem. We then used as upper bound 1.015 times the initial lower bound.) The average lower bound gap after the initial ascent (i.e. the difference between the optimal tour length and the lower bound as a percentage of the optimal tour length) was 0.13% for $n=60$, 0.31% for $n=80$ and 0.17% for $n=100$. These figures are 0.13% for $n=60$, 0.22% for $n=70$ and 0.12% for $n=80$ for the method of Smith and Thompson on their random table problems.

To decrease the actual computer time we eliminated edges at the end of the initial ascent in the following way. If fixing a certain edge into the minimal 1-tree causes the lower bound to exceed the current upper bound minus one, that edge can be eliminated. This complete operation can be performed efficiently in $O(n^2)$. To extend the sets of required and forbidden edges the same two simple rules are used as formulated in Section 4 (The branching scheme). Now the nodes are renumbered following the order in the best known solution after the initial ascent. By keeping the track of the first and last available edge for each city we can save time needed for the search of shortest distances in the algorithm for the minimal spanning 1-tree. For an average 60-node problem we eliminate about 90% of the edges. This approach also gives the opportunity to accelerate the program by using the Kruskal [11] or a similar algorithm instead of the actually used algorithm of Prim–Dijkstra [3,13].

Using this algorithm (coded in Fortran V) the mean time for finding a minimal 1-tree is about 12 milliseconds for a typical 60-node problem (Smith and Thompson: 61 milliseconds). Except for this subalgorithm the program is written in Pascal and run on a CDC Cyber 750 computer of the Academic Computer Centre of Amsterdam (SARA).

The most difficult Euclidean problem was a 60-node problem (R 605) that took for the depth-first version 46 seconds cpu time and 14 K memory words of 60 bits. For the breadth-first version the required number of central memory words was larger for some Euclidean problems: 21 K was the maximum. For the depth-first version the most difficult random table problem was a 100-node problem that took 222 seconds cpu time and 20 K memory words of 60 bits, for the breadth-first version a 80-node problem that took 60.3 seconds cpu time and 21 K memory words.

For the creation of the random table problems a Pascal version of the random generator routine ggubs from the IMSL (version Feb 1979) was used with random number seed $n^2 + \mathrm{nr}$ where n is the number of cities and nr the sequence number ($\mathrm{nr} = 1,2,\ldots,15$).

The random table problems with $n=100$ have not been solved by the breadth-first version as the computer facilities were quite scarce and the results of breadth- and depth-first are rather similar for $n=60$ and $n=80$.

7. Conclusions

We described an algorithm for the TSP that gives very good results for Euclidean problems and good results for random table problems. In our opinion the strongest parts of it are the branching scheme and the heuristic subalgorithm. With other Langrangean type algorithms this one has in common the rather arbitrary choice of some parameters.

The two versions of the algorithm, breadth-first and depth-first search, give approximately equivalent results. The larger cpu-time for the depth-first version is compensated by a smaller number of required central memory words.

One possible way to speed up the algorithm is to try for the reduction of the number of subsets. In our current research we aim to obtain this reduction by analyzing the best 1-tree in each subset. The first results are encouraging.

References

[1] N. Christofides, The shortest hamiltonian chain of a graph, SIAM J. Appl. Math. 19 (1970) 689–696.

[2] N. Christofides, Worst-case analysis of a new heuristic for the traveling salesman problem, Carnegie–Mellon University, Pittsburgh (1976).

[3] E.W. Dijkstra, A note on two problems in connection with graphs, Numer. Math. 1 (1959) 269–271.

[4] A.M. Geoffrion, Lagrangean relaxation for integer programming, Math. Programming Study 2 (1974) 81–114.
[5] K. Helbig Hansen and J. Krarup, Improvements of the Held–Karp algorithm for the symmetric traveling-salesman problem, Math. Programming (1974) 87–96.
[6] M. Held and R.M. Karp, A dynamic programming approach to sequencing problems, SIAM 10 (1962) 196–210.
[7] M. Held and R.M. Karp, The traveling-salesman problem and minimum spanning trees, Operations Res. 18 (1970) 1138–1162.
[8] M. Held and R.M. Karp, The traveling-salesman problem and minimum spanning trees: Part II, Math. Programming 1 (1971) 6–26.
[9] M. Held, P. Wolfe and H.P. Crowder, Validation of subgradient optimization, Math. Programming 6 (1974) 62–88.
[10] R.L. Karg and G.L. Thompson, A heuristic approach to solving traveling salesman problems, Management Sci. 10 (1964) 225–248.
[11] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proc. Amer. Math. Soc. 2 (1956) 48–50.
[12] S. Lin, Computer solutions of the traveling-salesman problem, BSTJ 44 (1965) 2245–2269.
[13] R.C. Prim, Shortest connection networks and some generalizations, BSTJ 36 (1957) 1389–1401.
[14] T.H.C. Smith and G.L. Thompson, A Lifo implicit enumeration search algorithm for the symmetric traveling salesman problem using Held and Karp's 1-tree relaxation, Ann. Discrete Math. 1 (1977) 479–493.