Alcom-FT Technical Report Series
ALCOMFT-TR-03-8

The Double-Tree Approximation for Metric TSP: Is the Best Good Enough? *

Vladimir Deineko[1] and Alexander Tiskin[2]

[1] Warwick Business School, The University of Warwick, Coventry CV47AL, UK
V.Deineko@warwick.ac.uk
[2] Dept. of Computer Science, The University of Warwick, Coventry CV47AL, UK
tiskin@dcs.warwick.ac.uk

Abstract. The Metric Travelling Salesman Problem (TSP) is a classical NP-hard optimisation problem. The double-tree heuristic allows one to approximate the solution of the Metric TSP within a factor of 2, by picking an arbitrary solution from a restricted space of solutions to the original problem. Such an approach raises two natural questions: can we find efficiently a solution that is optimal in the restricted space? will this optimal solution provide a better approximation ratio than an arbitrary solution? Paper [3, 4] answers the first question in the affirmative, by presenting an algorithm that finds the optimal double-tree solution to Metric TSP in time $O(2^d n^3)$ and space $O(n^2)$, where d is the maximum node degree in the corresponding minimal spanning tree (e.g. in the Euclidean case, $d \leq 6$). We improve these bounds to time $O(2^d n^2)$ and space $O(4^d n)$. We also give a partial answer to the second question, by presenting a lower bound of 1.666 on the approximation ratio of the optimal double-tree solution. Computational experiments show that the optimal double-tree algorithm is practical, providing one of the best known tour-constructing heuristics.

1 Introduction

The Metric Travelling Salesman Problem (TSP) is a classical combinatorial optimisation problem. We represent a set of n points in a metric space by a complete weighted graph on n nodes, where the weight of an edge is defined by the distance between the corresponding points. The objective of the Metric TSP is to find in this representing graph a Hamiltonian cycle of minimum weight.

The Metric TSP is well-known to be NP-hard, even for the standard Euclidean metric [11]. The general Metric TSP is also NP-hard to approximate to within a ratio 1.00456 ([13], see also [7]). There is no known polynomial-time approximation for Metric TSP with a ratio better than 1.5. On the other hand, paper [2] provides a polynomial-time approximation scheme (with the ratio arbitrarily close to 1) for Euclidean TSP, or more generally for Metric TSP with the metric defined by a Minkowski vector norm.

* Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

Two standard methods, double-tree [16] and Christofides' [5, 17] allow one to approximate efficiently the solution of the general Metric TSP within a factor of 2 and 1.5, respectively. Both methods build a weighted Eulerian graph on the given point set, select an Euler tour of the graph, and then perform *shortcutting* on this tour in order to remove repeated nodes. Thus, in both methods the weight of the shortest Hamiltonian cycle is approximated by picking an element from a restricted space of Hamiltonian cycles, namely the shortcuttings of a particular Euler tour. Note that the picked solution need not be optimal in the restricted space in order to achieve the claimed approximation factors. In fact, the Eulerian graph itself has weight at most a factor of 2 (respectively, 1.5) higher than the optimal TSP solution, and the shortcutting process cannot increase the weight. Hence, any shortcutting will provide the claimed approximation ratio.

A straightforward implementation of the double-tree method does not even need to construct an explicit Euler tour. It is sufficient to traverse the nodes of the minimum spanning tree in depth-first order, and to include the first appearance of each node in the Hamiltonian tour (see [10, 15]). We will refer to this algorithm variant as *the depth-first double-tree algorithm*. By incorporating some local optimisation features into the double-tree method one can obtain, for example, the nearest insertion heuristic or the nearest addition heuristic (see [10, 16]). It is not that straightforward to add local optimisation to the Christofides method, therefore most researchers have used a plain version of the method. However, recent attempts of Johnson and McGeoch [9] to incorporate elements of local search into the Christofides method appeared to be quite successful (see Section 4).

	Heuristics (by performance)	Wins	Quality
1	Saving (standard)	5	1.96
2	Saving (fast)	6	2.22
3	Saving (greedy)	5	2.75
4	Farthest Insertion (convex hull)	3	3.64
5	Random Insertion	3	4.26
6	Farthest Insertion	1	4.55
...	...	...	...
11	Cheapest Insertion	1	7.54
12	**Christofides**	**0**	**9.67**
...	...	...	...
37	**Double Tree (depth-first)**	**0**	**27.9**

Table 1. Results of Reinelt's experiment

Despite nice theoretical properties of the two methods, neither performed well in previous computational experiments (see [8, 15]). For instance, Reinelt [15] compared 37 tour constructing heuristics, including the depth-first double-tree and Christofides algorithms, on a set of 24 geometric instances from the TSPLIB database [14]. The results of Reinelt's experiment are summarised in

Table 1 (see also [15], p. 98). In this table, the column "Wins" shows, for each heuristic, the number of instances on which this particular heuristic obtained a better approximation than all the other heuristics. The column "Quality" shows the average percentage excess over the best solution found by the collection of all 37 heuristics. The depth-first version of the double-tree algorithm is at the bottom of the list. The Christofides heuristic is better, but still quite far from the top.

Intuitively, it is clear that the reason for the poor performance of the heuristics may be in the wrong decisions made in the shortcutting steps, especially if we take into account the number of possible choices that need to be made. It can be shown, in particular, that the number of tours that can be obtained by applying the double-tree method is exponential. Therefore, the following questions are interesting both from theoretical and practical perspective:

- can we find efficiently an optimal shortcutting of a given Eulerian graph?
- will the optimal shortcutting provide a better approximation than an arbitrary shortcutting?

For the Christofides method, paper [12] answers the first question in the negative, by showing that the problem of finding the optimal shortcutting of Christofides' Eulerian graph is NP-hard. For the double-tree method, paper [3, 4] answers that question in the affirmative, by presenting an algorithm that finds the optimal shortcutting of the double-tree graph in time $O(2^d n^3)$ and space $O(n^2)$, where d is the maximum node degree in the tree. In Section 2, we improve these bounds to time $O(2^d n^2)$ and space $O(4^d n)$. Note that in the Euclidean case $d \leq 6$, therefore we have a polynomial-time algorithm for Euclidean TSP. In what follows we refer to this algorithm as the optimal double-tree algorithm.

The second question is answered in the negative, both for the Christofides and the depth-first double-tree algorithms, by standard worst-case constructions providing a tight lower bound of 1.5, respectively 2, on the approximation ratio. The case of the optimal double-tree algorithm seems to be more difficult. In Section 3, we present a worst-case Euclidean TSP instance that provides a lower bound of 1.622 on the approximation ratio of the optimal double-tree algorithm. A simple modification of this construction provides a lower bound of 1.666 in the case of a general (not necessarily Euclidean) metric. The question of closing the gap between the above lower bounds and the upper bound of 2 remains open.

In Section 4 we provide results of computational experiments which show that the approximation quality of the optimal double tree algorithm is one of the best among known tour-constructing heuristics.

In the rest of this paper, we denote the distance between u and v by $d(u, v)$. For brevity, for a set A and an element a, we write $A \cup a$ instead of $A \cup \{a\}$, and $A \setminus a$ instead of $A \setminus \{a\}$.

2 The algorithm

Let G be a weighted graph representing the Metric TSP problem on n points. The double-tree method consists of the following stages:

- construct the minimum spanning tree of G;
- duplicate every edge of the tree, obtaining an n-node Eulerian graph;
- select an Euler tour of the double-tree graph;
- reduce the Euler tour to a Hamiltonian cycle by repeated *shortcutting*, i.e. replacing a node sequence a, b, c by a, c.

We say that a Hamiltonian cycle *conforms* to the doubled spanning tree, if it can be obtained from that tree by shortcutting one of its Euler tours.

In our optimal double-tree algorithm, we refine the bottom-up dynamic programming approach of [3, 4]. Initially, we select an arbitrary node r as the root of the tree. In the resulting rooted tree, the terms "parent", "child", "ancestor", "descendant" etc. have their usual meaning. For a node u, we denote by $C(u)$ the set of all children of u, and by $T(u)$ the maximal subtree rooted at u, i.e. the set of all descendants of u (including u itself). For a set of siblings U, we denote by $T(U)$ the (disjoint) union of all $T(u)$, $u \in U$. A characteristic property of a conforming Hamiltonian cycle is as follows: for every node u, upon entering the subtree $T(u)$, the cycle must visit all nodes of $T(u)$ before leaving the subtree (and will therefore never re-enter it). We will say that the cycle *sweeps* the subtree $T(u)$.

The algorithm proceeds by computing shortest conforming Hamiltonian paths in progressively increasing subtrees, beginning at the leaves and finishing with the whole tree $T(r)$. Let u be the root of the current subtree. Let $V \subseteq C(u)$, $a \in T(V)$. By $D_V^u(a)$ we denote the weight of the shortest conforming Hamiltonian path starting from u, sweeping $T(V)$, and finishing at a.

Consider the current subtree $T(u)$. Processing this subtree will involve computing the values $D_V^u(a)$ for all $V \subseteq C(u)$, $a \in T(V)$. In order to process the subtree, we need the corresponding values for all subtrees rooted at the children of u. More precisely, we need the values $D_W^v(a)$ for every child $v \in C(u)$, every subset $W \subseteq C(v)$, and every destination node $a \in T(W)$. We do not need any explicit information on grandchildren and lower descendants of u, therefore we will never store more than $2^d n$ values.

Given the current subtree $T(u)$, the values $D_V^u(a)$ are computed inductively for all sets V of children of u. The induction is on the size of the set V. The base of the induction is trivial: no values D_V^u exist when $V = \emptyset$.

For the inductive step, suppose we are given a non-empty set $V \cup v \subseteq C(u)$, $v \notin V$. By the inductive hypothesis, we have the values $D_V^u(a)$ for all $a \in T(V)$. First, we need to consider the paths that go beyond these currently known values by a single edge. Let $a \in T(C(u) \setminus V)$. By $D_V^u\langle a\rangle$ we denote the weight of the shortest conforming Hamiltonian path starting from u, sweeping $T(V)$, and finally jumping away from $T(V)$ to a. When $V = \emptyset$, we have $D_\emptyset^u\langle a\rangle = d(u, a)$. Otherwise, suppose the path exits $T(V)$ at node x. We have

$$D_V^u\langle a\rangle = \min_{x \in T(V)} \left[D_V^u(x) + d(x, a)\right] \tag{1}$$

(see Figure 1).

Our next goal is to compute the values $D_{V \cup v}^u(a)$ for all $a \in T(v)$. The main part of the computation consists in computing a set of auxiliary values $D_{V,W}^u(v)$,

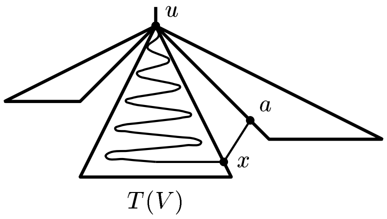

Fig. 1. Computation of $D_V^u\langle a\rangle$

for all subsets $W \subseteq C(v)$. Every such value represents the weight of the shortest conforming Hamiltonian path starting from u, sweeping $T(V)$, then sweeping $T(W)$, and finishing at v. If set W is empty, we have $D_{V,\emptyset}^u(v) = D_V^u(v)$. Otherwise, suppose the path enters $T(W)$ at node x. We have

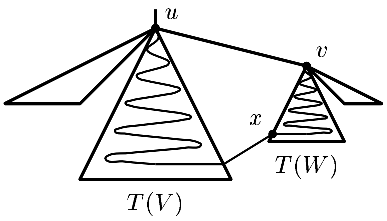

Fig. 2. Computation of $D_{V,W}^u(v)$

$$D_{V,W}^u(v) = \min_{x \in T(W)} \left[D_V^u\langle x\rangle + D_W^v(x)\right] \tag{2}$$

(see Figure 2). The required values $D_V^u\langle x\rangle$ have been computed in the current inductive step; the values $D_W^v(x)$ have been kept from the previous level of the tree. Note that the computed auxiliary set contains $D_{V\cup v}^u(v) = D_{V,C(v)}^u(v)$.

Now we can compute the values $D_{V\cup v}^u(a)$ for all $a \in T(v)$. A path corresponding to $D_{V\cup v}^u(a)$ must sweep $T(V)$, and then $T(v)$, before reaching a. While in $T(v)$, the path will first sweep a subtree $T(W)$, then visit node v, and then sweep the subtree $T(\overline{W})$. The weight of the path is determined by a disjoint bipartitioning $C(V) = W \cup \overline{W}$. If set W is empty, we have

$$D_{V\cup v}^u(a) = D_V^u(v) + D_{C(v)}^v(a) \tag{3}$$

(see Figure 3). Otherwise, we have

$$D_{V\cup v}^u(a) = \min_{W:\, x \in T(W),\, a \in T(\overline{W})} \left[D_{V,W}^u(v) + D_{\overline{W}}^v(a)\right] \tag{4}$$

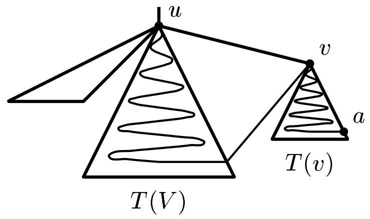

Fig. 3. Computation of $D^u_{V\cup v}(a)$, $a \in T(v)$, case $W = \emptyset$

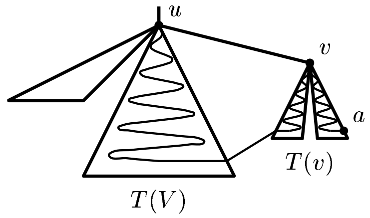

Fig. 4. Computation of $D^u_{V\cup v}(a)$, $a \in T(v)$, case $W \neq \emptyset$

(see Figure 4). The actual value of $D^u_{V\cup v}(a)$ is determined as the smaller of (3), (4).

We now have the values $D^u_{V\cup v}(a)$ for all $a \in T(v)$. By choosing a different node v to split off the current subset $V \cup v$, we can similarly compute the values $D^u_{V\cup v}(a)$ for all $a \in T(V \cup v)$.

The computation eventually reaches the root r of the tree, and establishes the values $D^r_S(a)$ for all $S \subseteq C(r)$ and all $a \neq r$. The weight of the optimal conforming path can now be determined as

$$\min_{a \neq r}\left[D^r_{C(r)}(a) + d(a,r)\right] \tag{5}$$

Theorem 1. *Given a set of n points in a metric space, the weight of the optimal conforming path can be computed in time $O(2^d n^2)$ and space $O(2^d n)$. The optimal path itself can be computed in the same asymptotic time, and in space $O(4^d n)$.*

Proof. The time complexity of the above algorithm is dominated by the computation of $D^u_V(a)$ in (1). As noted in [3,4], the total number of triples u, x, a is $O(n^2)$. For every such triple, the computation is performed 2^d times, therefore the overall time complexity is $O(2^d n)$. If our goal is the computation of the weight of the optimal path, at any given time we only need to keep the values $D^u_V(a)$ corresponding to the current node u and to its children (but not to any further descendants), therefore the space complexity is $O(2^d n)$.

In order to reconstruct the optimal cycle, we need to keep additionally the auxiliary values $D^u_{V,W}(v)$, obtained in the course of the computation for every parent-child pair u, v. The total amount of storage required for these values is $O(4^d n)$.

The optimal cycle is computed by solving recursively the following problem: given a node u, a set $V \subseteq C(u)$, and a node $a \in T(V)$, find the optimal path $P^u_V(a)$ starting from u, sweeping $T(V)$, and finishing at a. To compute the global optimal Hamiltonian cycle, it is sufficient to determine the path $P^r_{C(r)}(a)$, where r is the root of the tree, and a is the node that attains the minimum in (5).

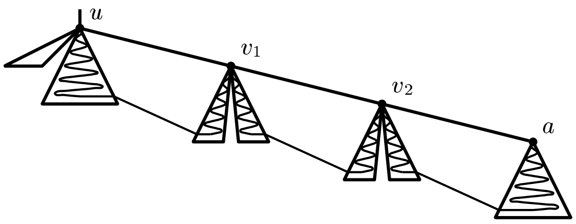

Fig. 5. Computation of $P^u_V(a)$, $a \in T(V)$

In order to compute the optimal path $P^u_V(a)$, let us consider the sequence of edges $u = v_0 \to v_1 \to v_2 \to \cdots \to v_k \to v_{k+1} = a$, joining nodes u and a in the tree (see Figure 5). For every node v_i between u and a, the path sweeps a subtree $T(W_i)$, then visits v_i, and then sweeps a subtree $T(\overline{W}_i)$. The weight of the path is determined by disjoint bipartitionings $C(v_i) \setminus v_{i+1} = W_i \cup \overline{W}_i$.

The optimal choice of bipartitionings can be found as follows. We construct a weighted directed graph with a source representing the subtree $T(C(u) \setminus v_1)$, a sink representing the subtree $T(C(a))$, and k intermediate layers of nodes representing the k subtrees $T(C(v_i) \setminus v_{i+1})$. Each layer consists of at most 2^{d-1} nodes, representing all different disjoint bipartitionings of the subtree. Every consecutive pair of layers (including the source and the sink) are fully connected by forward edges. In particular, the edge from a node representing the bipartitioning $C(v_i) \setminus v_{i+1} = X \cup \overline{X}$ to the node representing the bipartitioning $C(v_{i+1}) \setminus v_{i+2} = Y \cup \overline{Y}$, is weighted by the value $D^{v_i}_{\overline{X},Y}(v_{i+1})$. It is easy to see that an optimal choice of bipartitioning corresponds to the shortest path from source to sink in the representing graph. Such a shortest path can be found in time $O(4^d k)$ by the Bellman–Ford algorithm.

It now remains to apply the computation of the optimal path recursively in the subtrees $T(V \setminus v_1)$ and $T(a)$. The resulting time and space complexity of the algorithm are as claimed in the theorem statement. $\square$

The above algorithm finds the optimal solution to the Euclidean TSP in the restricted space of conforming Hamiltonian cycles. Paper [3, 4] proposes a further restriction of the solution space. There, admissible solutions are described by *PQ-trees*. A PQ-tree has *P-nodes*, which allow an arbitrary permutation of their children, and *Q-nodes*, which only allow a reversal. In our setting, the doubled minimum spanning tree is, in effect, a PQ-tree having only nodes of type P. To match the more general setting of [3, 4], it is not difficult to modify our algorithm to take care of the more restrictive Q-nodes.

3 The lower bounds

In the previous section we have developed an efficient implementation of the optimal double tree approximation for Metric TSP. We now address the question of the approximation ratio.

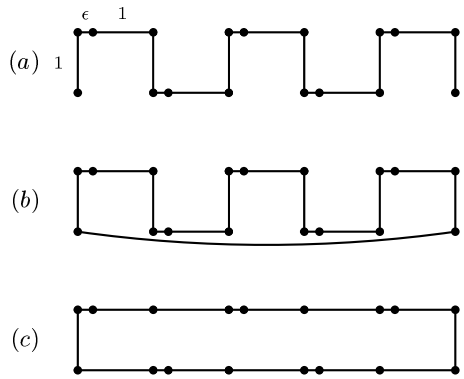

Fig. 6. Worst-case instance for the Christofides algorithm: (a) minimum spanning tree; (b) algorithm output; (c) optimal Hamiltonian cycle

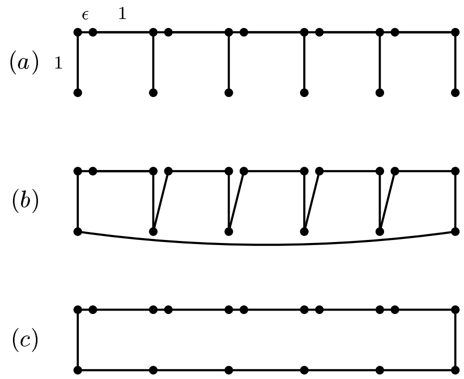

Fig. 7. Worst-case instance for the depth-first double-tree algorithm: (a) minimum spanning tree; (b) algorithm output; (c) optimal Hamiltonian cycle

As we have mentioned in the Introduction, standard Euclidean worst-case constructions exist for the Christofides [6] and the depth-first double-tree [10] algorithms. These constructions, shown in Figures 6, 7, provide a tight lower bound of 1.5, respectively 2, on the approximation ratio, independent of the choice of the shortcutting. Figures 6a, 7a show an instance point set and the (unique) minimum spanning tree. The vertical size of each instance set is 1, and the horizontal size is $n + o(n)$ (we assume that $\epsilon = o(1)$). In the case of the Christofides algorithm, the minimum spanning tree has exactly two odd-degree nodes, therefore the additional matching consists of a single edge. The resulting Eulerian graph (Figure 6b) is a cycle, hence no shortcutting is required. The weight of the cycle is $3n + o(n)$. In the case of the depth-first double-tree algorithm, the shortcutting of the doubled tree (Figure 7b) has weight $4n+o(n)$. For both algorithms, the optimal Hamiltonian cycle (Figures 6c, 7c) has weight $2n + o(n)$, therefore the approximation ratio on the given instance sets is 1.5 and 2, respectively. These lower bounds on the approximation ratio match the respective upper bounds.

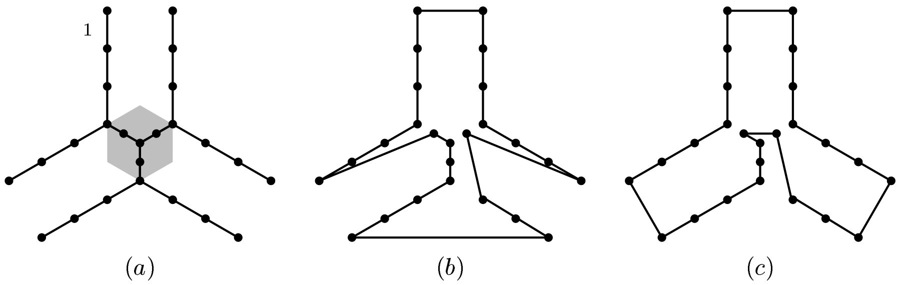

Fig. 8. Hard instance for the optimal double-tree algorithm: (a) minimum spanning tree; (b) algorithm output; (c) optimal Hamiltonian cycle

In contrast with the above two algorithms, no non-trivial lower bounds have been known, to the authors' knowledge, for the optimal double-tree algorithm. We propose a lower bound construction shown in Figure 8. The instance point set (Figure 8a) extends in each of three symmetric directions by $n + o(n)$. The optimal shortcutting of the doubled tree (Figure 8b) has weight $(8+\sqrt{3})n+o(n)$. The optimal Hamiltonian cycle (Figure 8c) has weight $6n + o(n)$, therefore the approximation ratio on the given instance set is $(8 + \sqrt{3})/6 \approx 1.622$. There remains a substantial gap between this lower bound and the upper bound of 2.

A better lower bound can be obtained from our construction in a general (not necessarily Euclidean) metric. Let us consider the same set of points in a metric defined by *the hexagonal norm* — a Minkowski vector norm with the unit disc in the shape of a regular hexagon (see Figure 8a). In this metric, the distance between two points is measured along a polygonal path composed from segments parallel to the edges of the unit disc. The minimum spanning tree and the shortest Hamiltonian cycle of our instance set remain unchanged in the new metric. However, the weight of the optimal shortcutting (Figure 8b) increases to $10n + o(n)$. Therefore, the improved lower bound on the approximation ratio is $10/6 \approx 1.666$.

4 Computational experiments

Previous to this work, computational experiments seemed to suggest that the practical performance of the double-tree method is not competitive with the other known tour constructing heuristics for the TSP. We implemented the optimal double-tree algorithm and tested its performance on various sets of test data. As it was expected, the introduction of optimal shortcutting had significantly improved the algorithm's quality.

We ran the optimal double tree algorithm on the same set of test instances as in the computational experiments by Reinelt, discussed in the Introduction (Table 1). Our results, presented in Table 2, suggest that the optimal double-tree algorithm appears to be the best tour-constructing heuristic.

	Heuristics (by performance)	Wins	Quality
1	**Double Tree (optimal)**	**20**	**0.39**
2	Saving (standard)	1	4.41
3	Saving (fast)	0	4.68
4	Saving (greedy)	0	5.22
5	Farthest Insertion (convex hull)	2	6.14
6	Random Insertion	1	6.77
7	Farthest Insertion	0	7.06
...	...	...	...
11	Cheapest Insertion	0	10.13
12	**Christofides**	**0**	**12.13**
...	...	...	...

Table 2. Revised position of the double tree algorithm in Reinelt's experiment

It is arguable, however, that the set of 24 test problems in the experiment may not be representative enough to make a judgement about the superiority of the optimal double tree algorithm. Website [1], created to support the recent DIMACS Implementation Challenge [9], provides an excellent opportunity for testing and evaluating new approaches to the TSP. The website contains a wide range of test instances and an updated picture of the state of the art in area of TSP heuristics. According to the results of the wide-range computational experiments conducted by Johnson and McGeoch [9], the Christofides heuristic with a local search component incorporated into the shortcutting step (row "Chr-G" in Table 3) appeared to be one of the best among tour constructing heuristics. The authors of [9] have also considered as a tour constructing heuristic a modification of the Cristofides algorithm, where the trees obtained during the process

	Uniform Instances			Clustered Instances			TSPLIB Instances		
Size	1000	3162	10K	1000	3162	10K	1000	3162	10K
CCA	10.11	11.47	11.73	7.42	8.90	10.99	10.64	11.56	11.14
Sav	11.38	11.78	11.82	13.51	15.97	17.21	11.96	12.14	10.85
Chr-G	9.80	9.79	9.81	8.08	9.01	9.21	8.72	9.41	8.86
Chr-HK	7.55	7.33	7.30	7.27	8.37	8.37	7.38	7.12	7.50
DT-Opt	7.36	7.82	8.02	6.86	8.34	8.34	7.40	8.33	8.26
DT4-Opt	6.48	6.82	6.91	5.77	7.04	7.34	6.60	7.18	7.48

Table 3. Average percentage excess over the Held and Karp bound (data on CCA, Sav, Chr-G, and Chr-HK are from [1])

of calculating the Held and Karp lower bounds are used as the input for further transformation into tours (row "Chr-HK" row in Table 3).

We also ran the optimal double-tree algorithm on the set of test instances from [1]. These are random uniform Euclidean instances, random clustered Euclidean distances, and instances from TSPLIB with 1000, 3162 and 10000 points. The results of our experiment are presented in Table 3. Row "DT-Opt" corresponds to the optimal double-tree algorithm. Row "DT4-Opt" contains results for a modification of the optimal double-tree algorithm, where the minimum spanning tree is pre-processed by increasing the degrees of some of the internal nodes. This transformation increases the space of the tours considered by the optimal double-tree algorithm. For comparison, Table 3 also includes the results for the saving heuristic (Sav), and for the convex hull and cheapest insertion heuristic (CCA).

The results in Table 3 clearly indicate the superiority of the optimal double-tree algorithm over other known tour constructing heuristics.

5 Conclusions

There were two reasons for us to work on improving the double-tree algorithm for the Euclidean TSP.

Firstly, this is an *exact* algorithm for finding an optimal solution in the exponential set of all tours that can be constructed by the double-tree heuristic. Although the double-tree heuristic has been well known for some time due to its theoretical properties (see e.g. [16]), the problem of finding the optimal solution remained open for about 20 years. Because of this "theory connection", we considered it as a challenge to make the exact algorithm as good as possible.

Secondly, computational experiments with the optimal double-tree algorithm show that the algorithm becomes one of the best tour constructing heuristics. The improvement in time complexity from $O(2^d n^3)$ to $O(2^d n^2)$ signifies a clear superiority of the optimal double tree over its main competitor — the greedy version of the Christofides algorithm. The improved space complexity will allow one to solve instances of sizes far beyond 10000, which was the largest size considered in our computational experiment.

References

1. DIMACS TSP challenge. `http://www.research.att.com/~dsj/chtsp`.
2. S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Journal of the ACM*, 45:753–782, 1998.
3. R. E. Burkard, V. G. Deineko, and G. J. Woeginger. The travelling salesman and the PQ-tree. *Mathematics of Operations Research*, 23(3):613–623, August 1998.
4. R. E. Burkard, V. G. Deineko, and G. J. Woeginger. Erratum: The travelling salesman and the PQ-tree. *Mathematics of Operations Research*, 24(1):262–272, February 1999.
5. N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon University, 1976.
6. G. Cornuejols and G. L. Nemhauser. Tight bounds for Christofides' travelling salesman heuristic. *Mathematical Programming*, 14:116–121, 1978.
7. L. Engebretsen. An explicit lower bound for TSP with distances one and two. *Algorithmica*, 35(4):301–319, 2003.
8. D. S. Johnson and L. A. McGeoch. The traveling salesman problem: a case study. In G. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimisation*, chapter 8, pages 215–310. John Wiley & Sons, 1997.
9. D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. P. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, chapter 9, pages 369–487. Kluwer Academic Publishers, 2002.
10. D. S. Johnson and C. H. Papadimitriou. Performance guarantees for heuristics. In E . L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The travelling salesman problem*, chapter 5, pages 145–180. Wiley, 1985.
11. C. H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, 4:237–247, 1977.
12. C. H. Papadimitriou and U. V. Vazirani. On two geometric problems related to the travelling salesman problem. *Journal of Algorithms*, 5:231–246, 1984.
13. C. H. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem. In *Proceedings of the 32nd ACM STOC*, pages 126–133, 2000. Updated version at `http://www.cs.berkeley.edu/~christos`.
14. G. Reinelt. TSPLIB — a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.
15. G. Reinelt. *The Travelling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag, 1994.
16. D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal of Computing*, 6:563–581, 1977.
17. A. Serdyukov. On some extremal walks in graphs. *Upravlyaemye systemy*, 17:76–79, 1978.