The Traveling Salesman Problem with Few Inner Points*

Vladimir G. Deĭneko[†] Michael Hoffmann[‡] Yoshio Okamoto[§]
Gerhard J. Woeginger[¶]

Abstract

We propose two algorithms for the planar Euclidean traveling salesman problem. The first runs in $O(k!kn)$ time and $O(k)$ space, and the second runs in $O(2^k k^2 n)$ time and $O(2^k kn)$ space, where n denotes the number of input points and k denotes the number of points interior to the convex hull.

Keywords: Exact algorithm, Fixed-parameter algorithm, Parameterization, Traveling salesman.

1 Introduction

In the *traveling salesman problem* (TSP) we are given n cities $1, 2, \ldots, n$ together with all the pairwise distances $d(i, j)$ between cities i and j. The goal is to find the shortest tour that visits every city exactly once and in the end returns to its starting city. The TSP is one of the most famous problems in combinatorial optimization, and it is well-known to be NP-hard. For more information on the TSP, the reader is referred to the book by Lawler, Lenstra, Rinnooy Kan & Shmoys [14].

A special case of the TSP is the so-called *Euclidean TSP*, where the cities are points in the Euclidean plane, and the distances are simply the Euclidean distances. A special case of the Euclidean TSP is the *convex Euclidean TSP*, where the cities are further restricted so that they lie in convex position. The Euclidean TSP is still NP-hard [9, 17], but the convex Euclidean TSP is quite easy to solve: Running along the boundary of the convex hull yields a shortest tour. Motivated by these two facts, we pose the following natural question: What is the influence of the number of inner points on the complexity of the problem? Here, an *inner point* of a finite point set P is a point from P which lies in the interior of the convex hull of P. Intuitively, we might say that "Fewer inner points make the problem easier to solve."

*The extended abstract version of this paper has appeared in Proceedings of 10th International Computing and Combinatorics Conference (COCOON 2004) [4]. Vladimir Deĭneko and Gerhard Woeginger are supported by the NWO project 613.000.322 "Exact Algorithms," and Yoshio Okamoto is supported by the Berlin-Zurich Joint Graduate Program "Combinatorics, Geometry, and Computation" (CGC), financed by ETH Zurich and the German Science Foundation (DFG). The authors are also grateful to Emo Welzl for helpful discussions.

†Warwick Business School, The University of Warwick, Conventry CV4 7AL, United Kingdom. E-mail: `v.deineko@warwick.ac.uk`.

‡Institute of Theoretical Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland. E-mail: `hoffmann@inf.ethz.ch`.

§Corresponding author. Institute of Theoretical Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland. E-mail: `okamotoy@inf.ethz.ch`.

¶Department of Mathematics and Computer Science, Techical University of Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. E-mail: `gwoegi@win.tue.nl`.

In the following, we answer this question and support the intuition above by providing simple exact algorithms based on the dynamic programming paradigm. The precise statement is as follows.

Theorem 1. *The special case of the Euclidean TSP with few inner points can be solved in the following time and space complexity. Here, n denotes the total number of cities and k denotes the number of cities in the interior of the convex hull. 1. In time $\mathrm{O}(k!kn)$ and space $\mathrm{O}(k)$. 2. In time $\mathrm{O}(2^k k^2 n)$ and space $\mathrm{O}(2^k kn)$.*

Here, we assume that the convex hull of a given point set is already determined, which can be done in time $\mathrm{O}(n \log n)$ and space $\mathrm{O}(n)$. Further, note that the above space bounds do not count the space needed to store the input but they just count the space in working memory (as usual in theoretical computer science).

Theorem 1 implies that, from the viewpoint of parameterized complexity [7, 16], these algorithms are fixed-parameter algorithms, when the number k of inner points is taken as a parameter, and hence the problem is fixed-parameter tractable (FPT). (A *fixed-parameter algorithm* has running time $\mathrm{O}(f(k)\mathrm{poly}(n))$, where n is the input size, k is a parameter and $f : \mathbb{N} \to \mathbb{N}$ is an arbitrary computable function. For example, an algorithm with running time $\mathrm{O}(440^k n)$ is a fixed-parameter algorithm whereas one with $\mathrm{O}(n^k)$ is not.) Observe that the second algorithm gives a polynomial-time exact solution to the problem when $k = \mathrm{O}(\log n)$.

Related work Since the literature on the TSP and its variants is vast, we only point out studies on the TSP itself which are closely related to our result. To the authors' knowledge, only few papers studied the parameterized complexity of the Euclidean TSP. Probably the most closely related one is a paper by Deĭneko, van Dal & Rote [5]. They studied the Euclidean TSP where the inner points lie on a line. The problem is called the *convex-hull-and-line TSP*. They gave an algorithm running in $\mathrm{O}(kn)$ time, where k is the number of inner points. Deĭneko & Woeginger [6] studied a slightly more general problem called the *convex-hull-and-ℓ-line TSP*, and gave an algorithm running in $\mathrm{O}((k/\ell)^\ell n^2)$ time. Compared to these results, our algorithms deal with the most general situation, and are still fixed-parameter algorithms with respect to k. As for approximation algorithms, Arora [1] and Mitchell [15] found polynomial-time approximation schemes (PTAS) for the Euclidean TSP. Rao & Smith [18] gave a PTAS with better running time $\mathrm{O}(n \log n + 2^{\mathrm{poly}(1/\varepsilon)} n)$. As for exact algorithms, Held & Karp [11] and independently Bellman [3] provided a dynamic programming algorithm to solve the TSP optimally in $\mathrm{O}(2^n n^2)$ time and $\mathrm{O}(2^n n)$ space. For the Euclidean TSP, Hwang, Chang & Lee [13] gave an algorithm to solve in $n^{\mathrm{O}(\sqrt{n})}$ time based on the so-called separator theorem.

Organization The next section gives a fundamental property of optimal tours. Sect. 3 and 4 describe our two algorithms for the Euclidean TSP. We conclude with some extensions and open problems at the final section.

2 Fundamental property

Let P be a finite point set on the Euclidean plane. A point $p \in P$ is called an *inner* point if p lies in the interior of the convex hull of P. We denote by $\mathsf{Inn}(P)$ the set of inner points of P. A point $p \in P$ is called an *outer* point if it is not an inner point, i.e., it is on the boundary of the convex hull of P. We denote by $\mathsf{Out}(P)$ the set of outer points of P. Let $n := |P|$ and $k := |\mathsf{Inn}(P)|$. (So, we have $|\mathsf{Out}(P)| = n - k$.)

A *tour* on P is a linear order $(x_1, x_2, \ldots, x_n)$ on P. We say that this tour *starts at* x_1. We often identify the tour $(x_1, \ldots, x_n)$ on P with a closed polygonal curve consisting of the line segments $\overline{x_1x_2}$, $\overline{x_2x_3}$, $\ldots$, $\overline{x_{n-1}x_n}$, $\overline{x_nx_1}$. The *length* of the tour is the Euclidean length of this polygonal curve, i.e., $\sum_{i=1}^{n-1} d(x_i, x_{i+1}) + d(x_n, x_1)$, where $d(x_i, x_{i+1})$ stands for the Euclidean distance from x_i to x_{i+1}.

A well-known result by Flood [8] states that a shortest tour for a point set in the Euclidean plane cannot cross itself. A consequence of this result (used, for instance, by Deĭneko, van Dal & Rote [5]) is that a shortest tour must traverse the points on $\mathsf{Out}(P)$ in a cyclic order. (Here, we call a linear order on $\mathsf{Out}(P)$ *cyclic* if every two consecutive points in the order are also consecutive on the boundary of the convex hull of P.)

With this observation, we can establish the following naive algorithm to compute a shortest tour: Take an arbitrary cyclic order on $\mathsf{Out}(P)$, then look through all tours (i.e., the linear orders) π on P which respect this cyclic order. (Here, for a set S and a subset $S' \subseteq S$, we say a linear order π on S *respects* a linear order π' on S' if the restriction of π onto S' is π'.) Compute the length of each tour, and output the best one among them. The number of such tours is $\mathrm{O}(k!n^k)$. Therefore, the running time of this algorithm is $\mathrm{O}(k!n^{k+1})$. So, if k is constant, this algorithm runs in polynomial time. However, it is not a fixed-parameter algorithm with respect to k since k appears in the exponent of n.

3 First algorithm

Our first algorithm uses the following idea. We first fix a cyclic order γ on $\mathsf{Out}(P)$, and look through all linear orders π on $\mathsf{Inn}(P)$. Then, we find a shortest tour on P which respects both γ on $\mathsf{Out}(P)$ and π on $\mathsf{Inn}(P)$, and output a minimum one over all choices of π. Later we will show that we can compute such a tour in $\mathrm{O}(kn)$ time and $\mathrm{O}(k)$ space for each π. Then, since the number of linear orders on $\mathsf{Inn}(P)$ is $k!$ and they can be enumerated in amortized $\mathrm{O}(1)$ time per one with $\mathrm{O}(k)$ space [19], overall the algorithm runs in $\mathrm{O}(k!kn)$ time and $\mathrm{O}(k)$ space.

Now, given a cyclic order $\gamma = (p_1, \ldots, p_{n-k})$ on $\mathsf{Out}(P)$ and a linear order $\pi = (q_1, \ldots, q_k)$ on $\mathsf{Inn}(P)$, we describe how to compute a shortest tour among those which respect γ and π by dynamic programming.

We consider a three-dimensional array $F_1[i, j, m]$, where $i \in \{1, \ldots, n-k\}$, $j \in \{0, 1, \ldots, k\}$, and $m \in \{\mathsf{Inn}, \mathsf{Out}\}$. The first index i represents the point p_i in $\mathsf{Out}(P)$, the second index j represents the point q_j in $\mathsf{Inn}(P)$, and the third index m represents the position. The value $F_1[i, j, m]$ represents the length of a shortest *path* on $\{p_1, \ldots, p_i\} \cup \{q_1, \ldots, q_j\}$ that satisfies the following four conditions. (1) It starts at $p_1 \in \mathsf{Out}(P)$. (2) It visits the points in $\{p_1, \ldots, p_i\} \cup \{q_1, \ldots, q_j\}$ exactly once. (If $j = 0$, set $\{q_1, \ldots, q_j\} = \emptyset$.) (3) It respects the orders γ and π. (4) If $m = \mathsf{Out}$, then it ends at p_i (an outer point). If $m = \mathsf{Inn}$, then it ends at q_j (an inner point).

Then, the length of a shortest tour respecting π and γ can be computed as

$$\min\{F_1[n-k, k, \mathsf{Out}] + d(p_{n-k}, p_1), F_1[n-k, k, \mathsf{Inn}] + d(q_k, p_1)\}.$$

Therefore, it suffices to know the values $F_1[i, j, m]$ for all possible i, j, m.

To do that, we establish a recurrence. First let us look at the boundary cases. (1) Since we start at p_1, set $F_1[1, 0, \mathsf{Out}] = 0$. (2) There are some impossible states for which we set the values to ∞. Namely, for every $j \in \{1, \ldots, k\}$ set $F_1[1, j, \mathsf{Out}] = \infty$, and for every $i \in \{1, \ldots, n-k\}$ set $F_1[i, 0, \mathsf{Inn}] = \infty$.

Now, assume we want to visit the points of $\{p_1, \ldots, p_i\} \cup \{q_1, \ldots, q_j\}$ while respecting the orders γ and π and arrive at q_j. How can we get to this state? Since we respect the orders, either (1) first we have to visit the points of $\{p_1, \ldots, p_i\} \cup \{q_1, \ldots, q_{j-1}\}$ to arrive at p_i then move to q_j, or (2) first we have to visit the points of $\{p_1, \ldots, p_i\} \cup \{q_1, \ldots, q_{j-1}\}$ to arrive at q_{j-1} then move to q_j. Therefore, we have

$$F_1[i, j, \mathsf{Inn}] = \min\{F_1[i, j-1, \mathsf{Out}] + d(p_i, q_j), F_1[i, j-1, \mathsf{Inn}] + d(q_{j-1}, q_j)\} \tag{1}$$

for $(i,j) \in \{1,\ldots,n{-}k\} \times \{1,\ldots,k\}$. Similarly, we have

$$F_1[i,j,\mathsf{Out}] = \min\{F_1[i{-}1,j,\mathsf{Out}] + d(p_{i-1},p_i), F_1[i{-}1,j,\mathsf{Inn}] + d(q_j,p_i)\} \tag{2}$$

for $(i,j) \in \{2,\ldots,n{-}k\} \times \{0,\ldots,k\}$, where $d(q_0,p_i)$ is considered ∞ for convenience. Since what is referred to in the right-hand sides of Equations (1) and (2) has smaller indices, we can solve this recursion in a bottom-up way by dynamic programming. This completes the dynamic-programming formulation for the computation of $F_1[i,j,m]$.

The size of the array is $(n{-}k) \times (k{+}1) \times 2 = \mathrm{O}(kn)$, and the computation of each entry can be done by looking up at most two other entries of the array. Therefore, we can fill up the array in $\mathrm{O}(kn)$ time and $\mathrm{O}(kn)$ space.

Now, we describe how to reduce the space requirement to $\mathrm{O}(k)$. For each $(i,j) \in \{1,\ldots,n{-}k\} \times \{1,\ldots,k\}$, consider when $F_1[i,j,\mathsf{Inn}]$ is looked up throughout the computation. It is looked up only when we compute $F_1[i,j{+}1,\mathsf{Inn}]$ and $F_1[i{+}1,j,\mathsf{Out}]$. So the effect of $F_1[i,j,\mathsf{Inn}]$ is local. Similarly, the value $F_1[i,j,\mathsf{Out}]$ is looked up only when we compute $F_1[i,j{+}1,\mathsf{Inn}]$ and $F_1[i{+}1,j,\mathsf{Out}]$. We utilize this locality in the computation.

We divide the computation process into some phases. For every $i \in \{1,\ldots,n{-}k\}$, in the i-th phase, we compute $F_1[i,j,\mathsf{Inn}]$ and $F_1[i,j,\mathsf{Out}]$ for all $j \in \{0,\ldots,k\}$. Within the i-th phase, the computation starts with $F_1[i,1,\mathsf{Out}]$ and proceeds along $F_1[i,2,\mathsf{Out}], F_1[i,3,\mathsf{Out}],\ldots$ until we reach $F_1[i,k,\mathsf{Out}]$. Then, we start calculating $F_1[i,1,\mathsf{Inn}]$ and proceed along $F_1[i,2,\mathsf{Inn}], F_1[i,3,\mathsf{Inn}],\ldots$ until we reach $F_1[i,k,\mathsf{Inn}]$. From the observation above, all the computation in the i-th phase only needs the outcome from the $(i{-}1)$-st phase and the i-th phase itself. Therefore, we only have to store the results from the $(i{-}1)$-st phase for each i. This requires only $\mathrm{O}(k)$ storage.

Thus, we have obtained Theorem 1.1.

4 Second algorithm

To obtain a faster algorithm, we make use of the trade-off between the time complexity and the space complexity. The idea of trade-off is also taken by the dynamic programming algorithm for the general traveling salesman problem due to Bellman [3] and Held & Karp [11], and our second algorithm is essentially a generalization of their algorithms. (For a nice exposition of this "dynamic programming across the subsets" technique together with other methods for exact computation, see Woeginger's survey article [20].)

In the second algorithm, first we fix a cyclic order $\gamma = (p_1,\ldots,p_{n-k})$ on $\mathsf{Out}(P)$. Then, we immediately start the dynamic programming. This time, we consider the following three-dimensional array $F_2[i,S,r]$, where $i \in \{1,\ldots,n{-}k\}$, $S \subseteq \mathsf{Inn}(P)$, and $r \in S \cup \{p_i\}$. We interpret $F_2[i,S,r]$ as the length of a shortest path on $\{p_1,\ldots,p_i\} \cup S$ that satisfies the following four conditions. (1) It starts at $p_1 \in \mathsf{Out}(P)$. (2) It visits the points in $\{p_1,\ldots,p_i\} \cup S$ exactly once. (3) It respects the order γ. (4) It ends at r.

Then, the length of a shortest tour can be computed as

$$\min\{F_2[n{-}k,\mathsf{Inn}(P),r] + d(r,p_1) \mid r \in \mathsf{Inn}(P) \cup \{p_{n-k}\}\}.$$

Therefore, it suffices to know the values $F_2[i,S,r]$ for all possible triples (i,S,r).

To do that, we establish a recurrence. The boundary cases are as follows. (1) Since we start at p_1, set $F_2[1,\emptyset,p_1] = 0$. (2) Set $F_2[1,S,p_1] = \infty$ when $S \neq \emptyset$, since this is an unreachable situation.

Let $i \in \{2,\ldots,n{-}k\}$ and $S \subseteq \mathsf{Inn}(P)$. We want to visit the points of $\{p_1,\ldots,p_i\} \cup S$ while respecting the order γ and arrive at p_i. How can we get to this state? Since we respect the order γ, we first have to visit the points of $\{p_1, \ldots, p_{i-1}\} \cup S$ to arrive at a point in $S \cup \{p_{i-1}\}$ and then move to p_i. Therefore, we have

$$F_2[i, S, p_i] = \min\{F_2[i-1, S, t] + d(t, p_i) \mid t \in S \cup \{p_{i-1}\}\} \tag{3}$$

for $i \in \{2, \ldots, n-k\}$ and $S \subseteq \mathsf{Inn}(P)$. Similarly, we have

$$F_2[i, S, r] = \min\{F_2[i, S \setminus \{r\}, t] + d(t, r) \mid t \in (S \setminus \{r\}) \cup \{p_i\}\} \tag{4}$$

for $i \in \{2, \ldots, n-k\}$, $S \subseteq \mathsf{Inn}(P)$, $S \neq \emptyset$ and $r \in S$. This completes the dynamic-programming formulation for the computation of $F_2[i, S, r]$.

The size of the array in this algorithm is $(n-k) \sum_{s=0}^{k} \binom{k}{s}(s+1) = \mathrm{O}(2^k kn)$, and the computation of each entry can be done by looking up $\mathrm{O}(k)$ other entries. Therefore, we can fill up the array in $\mathrm{O}(2^k k^2 n)$ time and in $\mathrm{O}(2^k kn)$ space. Thus, we obtain Theorem 1.2.

5 Concluding remarks

We have investigated the influence of the number of inner points in a two-dimensional geometric problem. Our result supports the intuition "Fewer inner points make the problem easier to solve," and nicely "interpolates" triviality when we have no inner point and intractability for the general case. This interpolation has been explored from the viewpoint of parameterized complexity. This kind of "distance-from-triviality" approach for parameterization has been recently further investigated by Guo, Hüffner & Niedermeier [10].

Let us remark that the result in this paper can also be applied to the two-dimensional Manhattan traveling salesman problem, where the distance is measured by the ℓ_1-norm. That is because the non-crossing property due to Flood [8] (mentioned in Sect. 2) also holds for that case. More generally, our algorithms solve any TSP instance (not necessarily geometric) for which $n-k$ points have to be visited in a specified order.

Furthermore, the ideas in this paper can be used to solve some variants of the TSP such as the prize-collecting TSP and the partial TSP in the Euclidean setup. In the Euclidean *prize-collecting TSP*, introduced by Balas [2], we are given an n-point set P in the Euclidean plane with a distinguished point $h \in P$, and a non-negative number $c(p) \in \mathbb{R}$ for each point $p \in P$ which we call the *penalty* of p. The goal is to find a subset $P' \subseteq P \setminus \{h\}$ and a tour on $P' \cup \{h\}$ starting at h which minimizes the length of the tour minus the penalties over all $p \in P' \cup \{h\}$. In the Euclidean *ℓ-partial TSP*, we are given an n-point set P on the Euclidean plane with a distinguished point $h \in P$, and a positive integer $\ell \leq n$. We are asked to find a shortest tour starting from h and consisting of ℓ points from P. (Usually the problem is called the k-partial TSP. However, since k is reserved for the number of inner points here, we use ℓ instead of k.) For these two problems, we can devise similar algorithms which run in polynomial time when $k = \mathrm{O}(\log n)$.

To the authors' knowledge, this is the first paper that deals with parameterized problems in terms of the number of inner points. Study of other geometric problems within this parameterized framework is an interesting direction of research. In a sister paper [12], the minimum weight triangulation problem was recently studied, and a fixed-parameter algorithm with respect to the number of inner points was provided.

A major open question is to improve the time complexity $\mathrm{O}(2^k k^2 n)$. For example, is there a polynomial-time algorithm for the Euclidean TSP when $k = \mathrm{O}(\log^2 n)$?

References

[1] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems, Journal of the ACM **45** (1998) 753–782.

[2] E. Balas. The prize collecting traveling salesman problem, Networks **19** (1989) 621–636.

[3] R. Bellman. Dynamic programming treatment of the traveling salesman problem, Journal of the ACM **9** (1962) 61–63.

[4] V.G. Deĭneko, M. Hoffmann, Y. Okamoto and G.J. Woeginger. The traveling salesman problem with few inner points, Proceedings of 10th International Computing and Combinatorics Conference (COCOON 2004), Lecture Notes in Computer Science **3106** (2004) 268–277.

[5] V. Deĭneko, R. van Dal and G. Rote. The convex-hull-and-line traveling salesman problem: A solvable case, Information Processing Letters **59** (1996) 295–301.

[6] V. Deĭneko and G.J. Woeginger. The convex-hull-and-k-line traveling salesman problem, Information Processing Letters **59** (1996) 295–301.

[7] R.G. Downey and M.R. Fellows. Parameterized Complexity, Springer, 1999.

[8] M.M. Flood. Traveling-salesman problem, Operations Research **4** (1956) 61–75.

[9] M.R. Garey, R.L. Graham and D.S. Johnson. Some NP-complete geometric problems, Proceedings of 8th Annual ACM Symposium on Theory of Computing (STOC '76) 1976, pp. 10–22.

[10] J. Guo, F. Hüffner and R. Niedermeier. A structural view on parameterizing problems: Distance from triviality, Proceedings of 1st International Workshop on Parameterized and Exact Computation (IWPEC 2004), Lecture Notes in Computer Science **3162** (2004) 162–173.

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

[12] M. Hoffmann and Y. Okamoto. The minimum weight triangulation problem with few inner points, Proceedings of 1st International Workshop on Parameterized and Exact Computation (IWPEC 2004), Lecture Notes in Computer Science **3162** (2004) 200–212.

[13] R.Z. Hwang, R.C. Chang and R.C.T. Lee. The searching over separators strategy to solve some NP-hard problems in subexponential time, Algorithmica **9** (1993) 398–423.

[14] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (eds.). The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, Wiley, 1985.

[15] J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric k-MST, TSP, and related problems, SIAM Journal on Computing **28** (1999) 1298–1309.

[16] R. Niedermeier. Invitation to fixed-parameter algorithms, Habilitation Thesis, Universität Tübingen, 2002.

[17] C.H. Papadimitriou. Euclidean TSP is *NP*-complete, Theoretical Computer Science **4** (1977) 237–244.

[18] S. Rao and W. Smith. Approximating geometric graphs via "spanners" and "banyans," Proceedings of 30th Annual ACM Symposium on Theory of Computing (STOC '98), 1998, pp. 540–550.

[19] R. Sedgewick. Permutation generation methods, ACM Computing Surveys **9** (1977) 137–164.

[20] G.J. Woeginger. Exact algorithms for NP-hard problems: A survey, Combinatorial Optimization — Eureka! You shrink!, Lecture Notes in Computer Science **2570** (2003) 185–207.