The Traveling Salesman Problem with Few Inner Points

Vladimir G. Deĭneko[1,*], Michael Hoffmann[2], Yoshio Okamoto[2,†], and Gerhard J. Woeginger[3,*]

[1] Warwick Business School, The University of Warwick, Conventry CV4 7AL, United Kingdom. orsvd@wbs.warwick.ac.uk

[2] Institute of Theoretical Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland. {hoffmann,okamotoy}@inf.ethz.ch

[3] Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. gwoegi@win.tue.nl

Abstract. We study the traveling salesman problem (TSP) in the 2-dimensional Euclidean plane. The problem is NP-hard in general, but trivial if the points are in convex position. In this paper, we investigate the influence of the number of inner points (i.e., points in the interior of the convex hull) on the computational complexity of the problem. We give two simple algorithms for this problem. The first one runs in $\mathrm{O}(k!kn)$ time and $\mathrm{O}(k)$ space, and the second runs in $\mathrm{O}(2^k k^2 n)$ time and $\mathrm{O}(2^k kn)$ space, when n is the total number of input points and k is the number of inner points. Hence, if k is taken as a parameter, this problem is fixed-parameter tractable (FPT), and also can be solved in polynomial time if $k = \mathrm{O}(\log n)$. We also consider variants of the TSP such as the prize-collecting TSP and the partial TSP in this setting, and show that they are FPT as well.

1 Introduction

The traveling salesman problem (TSP) is one of the most famous optimization problems, which comes along many kinds of applications such as logistics, scheduling, VLSI manufacturing, etc. In many practical applications, we have to solve TSP instances arising from the two-dimensional Euclidean plane, which we call the 2DTSP. Also most of the benchmark instances for TSP belong to this class. Theoretically speaking, the general 2DTSP is strongly NP-hard [9, 15]. On the other hand, the problem is trivial if the points are in convex position, i.e., they are the vertices of a convex polygon. Therefore, the following natural question is asked: 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."

In the following, we answer this question and support the intuition above by providing simple algorithms based on the dynamic programming paradigm. The first one runs in $\mathrm{O}(k!kn)$ time and $\mathrm{O}(k)$ space, and the second runs in $\mathrm{O}(2^k k^2 n)$ time and $\mathrm{O}(2^k kn)$ space, where n is the total number of input points and k is the number of inner points.

*Supported by the NWO project 613.000.322 "Exact Algorithms".

†Supported by the Berlin-Zurich Joint Graduate Program "Combinatorics, Geometry, and Computation" (CGC), financed by ETH Zurich and the German Science Foundation (DFG).

Hence, from the viewpoint of parameterized complexity [7, 14], these algorithms are fixed-parameter algorithms,[1] when the number of inner points is taken as a parameter, hence the problem is fixed-parameter tractable (FPT). Observe that the second algorithm gives a polynomial-time solution to the problem when $k = \mathrm{O}(\log n)$.

We also study two variants of the traveling salesman problem, both also strongly NP-hard: the prize-collecting traveling salesman problem, introduced by Balas [2], and the partial traveling salesman problem. We show that these problems on the Euclidean plane are FPT as well.

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 2DTSP. Probably the most closely related one is a paper by Deĭneko, van Dal and Rote [5]. They studied the 2DTSP 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 and Woeginger [6] studied a slightly more general problem called the *convex-hull-and-ℓ-line TSP*, and gave an algorithm running in $\mathrm{O}(k^\ell n^2)$ time. Compared to these results, our algorithms deal with the most general situation, and are fixed-parameter algorithms with respect to k. As for approximation algorithms, Arora [1] and Mitchell [13] found polynomial-time approximation schemes (PTAS) for the 2DTSP. Rao and Smith [16] gave a PTAS with better running time $\mathrm{O}(n \log n + 2^{\mathrm{poly}(1/\varepsilon)} n)$. As for exact algorithms, Held and Karp [10] 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 geometric problems, Hwang, Chang and Lee [12] gave an algorithm to solve the 2DTSP in $\mathrm{O}(n^{\mathrm{O}(\sqrt{n})})$ time based on the so-called separator theorem.

Organization The next section introduces the problem formally, and gives a fundamental lemma. Sect. 3 and 4 describe algorithms for the 2DTSP. Variations are discussed in Sect. 5. We conclude with an open problem at the final section.

2 Traveling salesman problem with few inner points

Let $P \subseteq \mathbb{R}^2$ be a finite point set in the Euclidean plane. The *convex hull* of P is the smallest convex set containing P. 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. Note that $P = \mathsf{Inn}(P) \cup \mathsf{Out}(P)$ and $\mathsf{Inn}(P) \cap \mathsf{Out}(P) = \emptyset$. Let $n := |P|$ and $k := |\mathsf{Inn}(P)|$. (So, we have $|\mathsf{Out}(P)| = n-k$.)

A *tour* on P is a linear order $(x_1, x_2, \ldots, x_n)$ of the points in 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 $\overline{x_1x_2}$, $\overline{x_2x_3}$, …, $\overline{x_{n-1}x_n}$, $\overline{x_nx_1}$. The *length* of the

[1] 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 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.

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}. The objective of the TSP is to find a shortest tour. The following lemma was probably first noted by Flood [8] and nowadays it is folklore.

Lemma 1 (Flood [8]). *Every shortest tour has no crossing.*

This lemma immediately implies the following lemma, which plays a fundamental role in our algorithm. We call a linear order on $\mathsf{Out}(P)$ *cyclic* if every two consecutive points in the order are also consecutive on the convex hull of P.

Lemma 2. *In every shortest tour on P, the points of $\mathsf{Out}(P)$ appear in a cyclic order.*

With Lemma 2, we can establish the following naive algorithm: take an arbitrary cyclic order on $\mathsf{Out}(P)$, then look through all tours (i.e., the linear orders) π on P which respect[2] this cyclic order. 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 fixed-parameter algorithm

First, let us remark that later on we always assume that, when P is given to an algorithm as input, $\mathsf{Out}(P)$ is distinguished together with a cyclic order $\gamma = (p_1, \ldots, p_{n-k})$. Also, note that the space complexity in the algorithms below do not count the input size, as usual in theoretical computer science.

Our first algorithm uses the following idea. We look through all linear orders on $\mathsf{Inn}(P)$. Let us first fix a linear order π on $\mathsf{Inn}(P)$. We will try to find a shortest tour on P which respects both the cyclic order γ on $\mathsf{Out}(P)$ and the linear order π on $\mathsf{Inn}(P)$. Then, we exhaust this procedure for all linear orders on $\mathsf{Inn}(P)$, and output a minimum one. Later we will show that we can compute such a tour in $\mathrm{O}(kn)$ time and $\mathrm{O}(k)$ space. 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 [17], overall the algorithm runs in $\mathrm{O}(k!kn)$ time and $\mathrm{O}(k)$ space.

Now, given a cyclic order γ on $\mathsf{Out}(P)$ and a linear order π on $\mathsf{Inn}(P)$, we describe how to compute a shortest tour among those which respect γ and π by dynamic programming. For dynamic programming in algorithmics, see the textbook by Cormen, Leiserson, Rivest and Stein [4], for example.

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 conditions.

[2] 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'.

- It starts at $p_1 \in \mathsf{Out}(P)$.
- It visits exactly the points in $\{p_1, \ldots, p_i\} \cup \{q_1, \ldots, q_j\}$. (If $j = 0$, set $\{q_1, \ldots, q_j\} = \emptyset$.)
- It respects the orders γ and π.
- 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.

- Since we start at p_1, set $F_1[1, 0, \mathsf{Out}] = 0$.
- 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

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

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

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

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 Equation (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 requires to look 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 start with $F_1[i, 1, \mathsf{Out}]$ and proceed along

$F_1[i, 2, \mathsf{Out}], F_1[i, 3, \mathsf{Out}], \ldots$, until we get $F_1[i, k, \mathsf{Out}]$. 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 get $F_1[i, k, \mathsf{Inn}]$. ¿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.

In this way, we obtain the following theorem.

Theorem 3. *The 2DTSP on n points including k inner points can be solved in $\mathrm{O}(k!kn)$ time and $\mathrm{O}(k)$ space. In particular, it can be solved in polynomial time if $k = \mathrm{O}(\log n / \log\log n)$.* □

4 Second fixed-parameter algorithm with better running time

To obtain a faster algorithm, we make use of the trade-off between the time complexity and the space complexity. Compared to the first algorithm, the second algorithm has a better running time $\mathrm{O}(2^k k^2 n)$ and needs a larger space $\mathrm{O}(2^k kn)$. 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 [10], and our second algorithm is essentially a generalization of their algorithms.

In the second algorithm, first we fix a cyclic order γ 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 conditions.

- It starts at $p_1 \in \mathsf{Out}(P)$.
- It visits exactly the points in $\{p_1, \ldots, p_i\} \cup S$.
- It respects the order γ.
- 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.

- Since we start at p_1, set $F_2[1, \emptyset, p_1] = 0$.
- Set $F_2[1, S, r] = \infty$ 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

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

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

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

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 $(n-k)\sum_{s=0}^{k} \binom{k}{s} s = \mathrm{O}(2^k kn)$, and the computation of each entry requires to look 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 the following theorem.

Theorem 4. *The 2DTSP on n points including k inner points can be solved in $\mathrm{O}(2^k k^2 n)$ time and $\mathrm{O}(2^k kn)$ space. In particular, it can be solved in polynomial time if $k = \mathrm{O}(\log n)$.* □

5 Variants of the traveling salesman problem

Since our approach to the TSP in the previous section is based on the general dynamic programming paradigm, it is also applicable to other variants of the TSP. In this section, we discuss two of them.

5.1 Prize-collecting traveling salesman problem

In the *prize-collecting TSP*, we are given an n-point set $P \subseteq \mathbb{R}^2$ with a distinguished point $h \in P$ called the *home*, 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 this section, the value of a tour (or a path) refers to the value of this objective.

For this problem, we basically follow the same procedure as the TSP, but a little attention has to be paid because in this case we have to *select* some of the points from P. In addition, we have to care about the position of h, in $\mathsf{Inn}(P)$ or in $\mathsf{Out}(P)$.

First algorithm First, let us consider the case $h \in \mathsf{Out}(P)$. In this case, we fix a cyclic order γ on $\mathsf{Out}(P)$, which starts with h, and we look through all linear orders on $\mathsf{Inn}(P)$. Let $\gamma = (p_1, p_2, \ldots, p_{n-k})$, where $p_1 = h$, and fix one linear order $\pi = (q_1, q_2, \ldots, q_k)$ on $\mathsf{Inn}(P)$. Then, 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 value $F_1[i, j, m]$ is interpreted as the value of an optimal path on $\{p_1, \ldots, p_i\} \cup \{q_1, \ldots, q_j\}$ which satisfies the following conditions.

- It starts at $p_1 \in \mathsf{Out}(P)$.
- It visits *some* points from $\{p_1, \ldots, p_i\} \cup \{q_1, \ldots, q_j\}$, and not more. (If $j = 0$, set $\{q_1, \ldots, q_j\} = \emptyset$.)
- It respects the orders γ and π.

– If $m = \mathsf{Out}$, then it ends at p_i, If $m = \mathsf{Inn}$, then it ends at q_j.

We want to compute the values $F_1[i, j, m]$ for all possible triples (i, j, m).

The boundary cases are: $F_1[1, j, \mathsf{Out}] = -c(p_1)$ for every $j \in \{1, \ldots, k\}$; and $F_1[i, 0, \mathsf{Inn}] = \infty$ for every $i \in \{1, \ldots, n-k\}$, and the main part of the recurrence is:

$$F_1[i, j, \mathsf{Inn}] = \min\{\min_{i' \in \{1,\ldots,i\}} (F_1[i', j-1, \mathsf{Out}] + d(p_{i'}, q_j) - c(q_j)), \\ \min_{j' \in \{0,\ldots,j-1\}} (F_1[i, j', \mathsf{Inn}] + d(q_{j'}, q_j) - c(q_j))\}$$

for $(i, j) \in \{1, \ldots, n-k\} \times \{1, \ldots, k\}$, and

$$F_1[i, j, \mathsf{Out}] = \min\{\min_{i' \in \{1,\ldots,i-1\}} (F_1[i', j, \mathsf{Out}] + d(p_{i'}, p_i) - c(p_i)), \\ \min_{j' \in \{0,\ldots,j\}} (F_1[i-1, j', \mathsf{Inn}] + d(q_{j'}, p_i) - c(p_i))\}$$

for $(i, j) \in \{2, \ldots, n-k\} \times \{0, \ldots, k\}$. For convenience, $d(q_0, p_i)$ is considered to be ∞.

Then, the value of an optimal prize-collecting tour respecting π and γ can be computed as

$$\min\{\min_{i \in \{1,\ldots,n-k\}} (F_1[i, k, \mathsf{Out}] + d(p_i, p_1)), \min_{j \in \{0,\ldots,k\}} (F_1[n-k, j, \mathsf{Inn}] + d(q_j, p_1))\}.$$

Since the size of the array is $\mathrm{O}(kn)$ and each entry can be filled by looking up $\mathrm{O}(n)$ other entries, the running time is $\mathrm{O}(kn^2)$. Therefore, looking through all linear orders on $\mathsf{Inn}(P)$, the overall running time of the algorithm is $\mathrm{O}(k!kn^2)$.

Next, let us consider the case $h \in \mathsf{Inn}(P)$. In this case, we look through all linear orders on $\mathsf{Inn}(P)$ which start with h, and also all cyclic orders on $\mathsf{Out}(P)$. Fix one linear order $\pi = (q_1, q_2, \ldots, q_k)$ on $\mathsf{Inn}(P)$, where $q_1 = h$, and one cyclic order $\gamma = (p_1, p_2, \ldots, p_{n-k})$ on $\mathsf{Out}(P)$. Then, we consider a three-dimensional array $F_1[i, j, m]$, where $i \in \{0, 1, \ldots, n-k\}$, $j \in \{1, \ldots, k\}$ and $m \in \{\mathsf{Inn}, \mathsf{Out}\}$. The interpretation and the obtained recurrence is similar to the first case, hence omitted. However, in this case, the number of orders we look through is $\mathrm{O}(k!n)$. Therefore, the overall running time of the algorithm in this case is $\mathrm{O}(k!kn^3)$. Thus, we obtain the following theorem.

Theorem 5. *The prize-collecting TSP in the Euclidean plane can be solved in* $\mathrm{O}(k!kn^3)$ *time and* $\mathrm{O}(kn)$ *space, when* n *is the total number of input points and* k *is the number of inner points. In particular, it can be solved in polynomial time if* $k = \mathrm{O}(\log n / \log\log n)$. □

Second algorithm Now we adapt the second algorithm for the 2DTSP to the prize-collecting TSP. Let us consider the case $h \in \mathsf{Out}(P)$. (The case $h \in \mathsf{Inn}(P)$ can be handled in the same way.) For a cyclic order $\gamma = (p_1, \ldots, p_{n-k})$ on $\mathsf{Out}(P)$ with $p_1 = h$, we define a 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 value of an optimal path on $\{p_1, \ldots, p_i\} \cup S$ that satisfies the following conditions.

- It starts at $p_1 \in \mathsf{Out}(P)$.
- It visits some points of $\{p_1, \ldots, p_i\} \cup S$.
- It respects the order γ.
- It ends at r.

Then, the value of an optimal tour can be computed as

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

The boundary cases are: $F_2[1, \emptyset, p_1] = -c(p_1)$; $F_2[1, S, r] = \infty$ when $S \neq \emptyset$. The main part of the recurrence is

$$F_2[i, S, p_i] = \min_{t \in S \cup \{p_{i-1}\}} (F_2[i-1, S, t] + d(t, p_i) - c(p_i))$$

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

$$F_2[i, S, r] = \min_{t \in (S \setminus \{r\}) \cup \{p_i\}} (F_2[i, S \setminus \{r\}, t] + d(t, r) - c(r))$$

for $i \in \{2, \ldots, n-k\}$, $S \subseteq \mathsf{Inn}(P)$ and $r \in S$. Then, we see that the computation can be done in $\mathrm{O}(2^k k^2 n)$ time and $\mathrm{O}(2^k kn)$ space.

Theorem 6. *The prize-collecting TSP in the Euclidean plane can be solved in* $\mathrm{O}(2^k k^2 n)$ *time and* $\mathrm{O}(2^k kn)$ *space, when* n *is the total number of input points and* k *is the number of inner points. In particular, it can be solved in polynomial time if* $k = \mathrm{O}(\log n)$. □

5.2 Partial traveling salesman problem

In the *ℓ-partial TSP*[3], we are given an n-point set $P \subseteq \mathbb{R}^2$ with a distinguished point $h \in P$ called the home, 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.

Because of space limitations, we do not give an adaptation of the first algorithm for the TSP, although it is possible but more tedious. So, we just describe a variation of the second algorithm.

Second algorithm Similarly to the prize-collecting TSP, we distinguish the cases according to the position of h, in $\mathsf{Inn}(P)$ or in $\mathsf{Out}(P)$. Here we only consider the case $h \in \mathsf{Out}(P)$. (The case $h \in \mathsf{Inn}(P)$ is similar.) Fix a cyclic order $\gamma = (p_1, \ldots, p_{n-k})$ on $\mathsf{Out}(P)$, where $p_1 = h$. We consider a four-dimensional array $F_2[i, S, r, m]$, where $i \in \{1, \ldots, n-k\}$, $S \subseteq \mathsf{Inn}(P)$, $r \in S \cup \{p_i\}$, and $m \in \{1, \ldots, \ell\}$. Then, $F_2[i, S, r, m]$ is interpreted as the length of a shortest path that satisfies the following conditions.

- It starts at $p_1 \in \mathsf{Out}(P)$.
- It visits exactly m points of $\{p_1, \ldots, p_i\} \cup S$.

[3] Usually the problem is called the k-partial TSP. However, since k is reserved for the number of inner points in the current work, we will use ℓ instead of k.

- It respects the order γ.
- It ends at r.

Note that the fourth index m represents the number of points which have already been visited. Then, the length of a shortest tour through ℓ points is

$$\min\{F_2[i, \mathsf{Inn}(P), r, \ell] + d(r, p_1) \mid i \in \{1, \ldots, n-k\}, r \in \mathsf{Inn}(P) \cup \{p_i\}\}.$$

Therefore, it suffices to compute the values $F_2[i, S, r, m]$ for all possible i, S, r, m.

The boundary cases are: $F_2[i, S, r, 1] = 0$ if $i = 1$ and $r = p_1$; otherwise $F_2[i, S, r, 1] = \infty$. Also, $F_2[1, S, p_1, m] = \infty$ for $m > 1$. The main part of the recurrence is:

$$F_2[i, S, p_i, m] = \min_{t \in S \cup \{p_{i-1}\}} (F_2[i-1, S, t, m-1] + d(t, p_i))$$

for $i \in \{2, \ldots, n-k\}$, $S \subseteq \mathsf{Inn}(P)$ and $m \in \{2, \ldots, \ell\}$;

$$F_2[i, S, r, m] = \min_{t \in (S \setminus \{r\}) \cup \{p_i\}} (F_2[i, S \setminus \{r\}, t, m-1] + d(t, r))$$

for $i \in \{1, \ldots, n-k\}$, $S \subseteq \mathsf{Inn}(P)$, $r \in S$ and $m \in \{2, \ldots, \ell\}$.

Although the size of the array is $\mathrm{O}(2^k k \ell n)$, we can reduce the space requirement to $\mathrm{O}(2^k k n)$ because of the locality with respect to the fourth index m. In this way, we obtain the following theorem.

Theorem 7. *The ℓ-partial TSP for the Euclidean plane can be solved in $\mathrm{O}(2^k k^2 \ell n)$ time and $\mathrm{O}(2^k k n)$ space, where n is the total number of input points and k is the number of inner points. In particular, it can be solved in polynomial time if $k = \mathrm{O}(\log n)$.*

6 Concluding remarks

We have investigated the influence of the number of inner points in a two-dimensional geometric problem. Our results support the intuition "Fewer inner points make the problem easier to solve," and nicely "interpolate" triviality when we have no inner point and intractability for the general case. This interpolation has been explored from the viewpoint of parameterized complexity. Let us remark that the results 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 Lemmata 1 and 2 are also true 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.

To the authors' knowledge, this work 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. Within this framework, the minimum weight triangulation problem was recently studied, and a fixed-parameter algorithm with respect to the number of inner points was given [11].

The 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 2DTSP when $k = \mathrm{O}(\log^2 n)$?

Acknowledgments We are grateful to Emo Welzl for helpful discussions.

References

1. S. Arora: Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. J. 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. J. ACM **9** (1962) 61–63.
4. T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein: Introduction to algorithms, 2nd edition. MIT Press, Cambridge, 2001.
5. V. Deĭneko, R. van Dal and G. Rote: The convex-hull-and-line traveling salesman problem: a solvable case, Inf. Process. Lett. **59** (1996) 295–301.
6. V. Deĭneko and G.J. Woeginger. The convex-hull-and-k-line traveling salesman problem. Inf. Process. Lett. **59** (1996) 295–301.
7. R.G. Downey and M.R. Fellows: Parameterized Complexity. Springer, Berlin, 1999.
8. M.M. Flood: Traveling salesman problem. Oper. Res. **4** (1956) 61–75.
9. M.R. Garey, R.L. Graham and D.S. Johnson: Some NP-complete geometric problems. Proc. 8th STOC, 1976, pp. 10–22.
10. M. Held and R. Karp: A dynamic programming approach to sequencing problems. J. SIAM **10** (1962) 196–210.
11. M. Hoffmann and Y. Okamoto: The minimum weight triangulation problem with few inner points. Submitted.
12. 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.
13. J. Mitchell: Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric k-MST, TSP, and related problems. SIAM J. Comput. **28** (1999) 1298–1309.
14. R. Niedermeier: Invitation to fixed-parameter algorithms. Habilitation Thesis, Universität Tübingen, 2002.
15. C.H. Papadimitriou: Euclidean TSP is *NP*-complete. Theor. Comput. Sci. **4** (1977) 237–244.
16. S. Rao and W. Smith: Approximating geometric graphs via "spanners" and "banyans." Proc. 30th STOC, 1998, pp. 540–550.
17. R. Sedgewick: Permutation generation methods. ACM Comput. Surveys **9** (1977) 137–164.