Fine-Grained Complexity Analysis of Two Classic TSP Variants*

Mark de Berg[1], Kevin Buchin[2], Bart M. P. Jansen[3], and Gerhard Woeginger[4]

1 **Eindhoven University of Technology, Eindhoven, The Netherlands**
m.t.d.berg@tue.nl

2 **Eindhoven University of Technology, Eindhoven, The Netherlands**
k.a.buchin@tue.nl

3 **Eindhoven University of Technology, Eindhoven, The Netherlands**
b.m.p.jansen@tue.nl

4 **Eindhoven University of Technology, Eindhoven, The Netherlands**
g.woeginger@tue.nl

Abstract

We analyze two classic variants of the TRAVELING SALESMAN PROBLEM using the toolkit of fine-grained complexity.

Our first set of results is motivated by the BITONIC TSP problem: given a set of n points in the plane, compute a shortest tour consisting of two monotone chains. It is a classic dynamic-programming exercise to solve this problem in $\mathcal{O}(n^2)$ time. While the near-quadratic dependency of similar dynamic programs for LONGEST COMMON SUBSEQUENCE and DISCRETE FRÉCHET DISTANCE has recently been proven to be essentially optimal under the Strong Exponential Time Hypothesis, we show that bitonic tours can be found in subquadratic time. More precisely, we present an algorithm that solves bitonic TSP in $\mathcal{O}(n \log^2 n)$ time and its bottleneck version in $\mathcal{O}(n \log^3 n)$ time. In the more general pyramidal TSP problem, the points to be visited are labeled $1, \ldots, n$ and the sequence of labels in the solution is required to have at most one local maximum. Our algorithms for the bitonic (bottleneck) TSP problem also work for the pyramidal TSP problem in the plane.

Our second set of results concerns the popular k-OPT heuristic for TSP in the graph setting. More precisely, we study the k-OPT decision problem, which asks whether a given tour can be improved by a k-OPT move that replaces k edges in the tour by k new edges. A simple algorithm solves k-OPT in $\mathcal{O}(n^k)$ time for fixed k. For 2-OPT, this is easily seen to be optimal. For $k = 3$ we prove that an algorithm with a runtime of the form $\widetilde{\mathcal{O}}(n^{3-\varepsilon})$ exists if and only if ALL-PAIRS SHORTEST PATHS in weighted digraphs has such an algorithm. For general k-OPT, it is known that a runtime of $f(k) \cdot n^{o(k/\log k)}$ would contradict the Exponential Time Hypothesis. The results for $k = 2, 3$ may suggest that the actual time complexity of k-OPT is $\Theta(n^k)$. We show that this is not the case, by presenting an algorithm that finds the best k-move in $\mathcal{O}(n^{\lfloor 2k/3 \rfloor + 1})$ time for fixed $k \geq 3$. This implies that 4-OPT can be solved in $\mathcal{O}(n^3)$ time, matching the best-known algorithm for 3-OPT. Finally, we show how to beat the quadratic barrier for $k = 2$ in two important settings, namely for points in the plane and when we want to solve 2-OPT repeatedly.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases Traveling salesman problem, fine-grained complexity, bitonic tours, k-opt

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.5

* This work was supported by NWO Gravitation grant 024.002.003 “Networks” (all authors), NWO grant 612.001.207 “A framework for progressive, user-steered algorithms in visual analytics” (Buchin), and NWO Veni grant “Frontiers in Parameterized Preprocessing” (Jansen).

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 5; pp. 5:1–5:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Motivation

We analyze two classic variants of the Traveling Salesman Problem (TSP) by applying the modern toolkit of fine-grained complexity analysis. The first TSP variant can for instance be found in Chapter 15 of the well-known textbook *"Introduction to Algorithms"* by Cormen, Leiserson, Rivest, and Stein [15]. The chapter discusses dynamic programming, and its problem section poses the following classic exercise:

> ***15-3 Bitonic euclidean traveling-salesman problem***
> In the ***euclidean traveling-salesman problem***, we are given a set of n points in the plane, and we wish to find the shortest closed tour that connects all n points. The general problem is NP-complete, and its solution is therefore believed to require more than polynomial time. J. L. Bentley has suggested that we simplify the problem by restricting our attention to ***bitonic tours***, that is, tours that start at the leftmost point, go strictly rightward to the rightmost point, and then go strictly leftward back to the starting point. In this case, a polynomial-time algorithm is possible. Describe an $\mathcal{O}(n^2)$-time algorithm for determining an optimal bitonic tour.

This exercise already showed up in the very first edition of the book in 1991. Since then, thousands of students pondered about it and (hopefully) found the solution. One might wonder whether $\mathcal{O}(n^2)$ runtime is best possible for this problem. As one of our main contributions, we will show that in fact it is not.

The second TSP variant concerns k-OPT, a popular local search heuristic that attempts to improve a suboptimal solution by a k-OPT *move* (or: k*-move* for short), which is an operation that removes k edges from the current tour and reconnects the resulting pieces into a new tour by inserting k new edges. The cases $k = 2$ [16] and $k = 3$ have been studied extensively with respect to various aspects such as experimental performance [7, 24, 27], (smoothed) approximation ratio [13, 26], rate of convergence [13, 17], and algorithm engineering [19, 21, 29, 30]. The decision problem associated with k-OPT asks, given a tour in an edge-weighted graph, whether it is possible to obtain a tour of smaller weight by replacing k edges. There are $\Theta(n^k)$ possibilities to choose k edges that leave the current tour, and for each choice the number of ways to reconnect the resulting pieces back into a tour is constant (for fixed k). As the weight change for each reconnection pattern can be evaluated in $\mathcal{O}(k)$ time, this simple algorithm finds the best k-OPT improvement in time $\mathcal{O}(n^k)$ for each fixed k. The survey chapter [25] by Johnson and McGeoch extensively discusses k-OPT. On page 233 they write:

> To complete our discussion of running times, we need to consider the time per move as well as the number of moves. This includes the time needed to *find* an improving move (or verify that none exists), together with the time needed to *perform* the move. In the worst case, 2-opt and 3-opt require $\Omega(n^2)$ and $\Omega(n^3)$ time respectively to verify local optimality, assuming all possible moves must be considered.

The two lower bounds in the last sentence are stated without further justification. It is clear that finding an improving k-move takes $\Omega(n^k)$ time, if we require that all possible moves must be enumerated *explicitly*. However, one might wonder whether there are other, faster algorithmic approaches that proceed without enumerating all moves. As one of our main contributions, we will show that such faster approaches do not exist for $k = 3$ (under the All-Pairs Shortest Paths conjecture), but do exist for all $k \geq 4$.

1.2 Our contributions

We investigate whether the long-standing runtimes of $\mathcal{O}(n^2)$ for bitonic tours and $\mathcal{O}(n^k)$ for finding k-opt improvements are optimal. Such optimality investigations usually involve two ingredients: fast algorithms and runtime lower bounds. While proving unconditional lower bounds is far out of reach, in recent years there has been an influx of techniques for establishing lower bounds on the running time of a given problem, based on a hypothesis about the best-possible running time for another problem. Recent results in this direction consider the problems of computing the Longest Common Subsequence [1, 10] of two length-n strings, the Edit Distance [5, 10] from one length-n string to another, or the Discrete Fréchet Distance [9] between two polygonal n-vertex curves in the plane. If one of these problems allows an algorithm with running time $\mathcal{O}(n^{2-\varepsilon})$, then this would yield an algorithm to test the satisfiability of an n-variable CNF formula ϕ in time $(2-\varepsilon)^n \cdot |\phi|^{\mathcal{O}(1)}$. As decades of research have not led to algorithms with such a running time for cnf-sat, this gives evidence that the classic $\mathcal{O}(n^2)$-time algorithms for these problems are optimal up to $n^{o(1)}$ factors.

Pyramidal tours in the plane. Consider a symmetric tsp instance that is defined by an edge-weighted complete graph. For a linear ordering $1, \ldots, n$ of the vertices in the graph, a *pyramidal* tour has the form $(1, i_1, \ldots, i_r, n, j_1, \ldots, j_{n-r-2})$, where $i_1 < i_2 < \ldots < i_r$ and $j_1 > j_2 > \ldots > j_{n-r-2}$. A *bitonic* tour for a Euclidean tsp instance is pyramidal with respect to the left-to-right order on the points in the plane. Bitonic and pyramidal tours play an important role in the combinatorial optimization literature on the tsp; see [6, 11, 20]. They form an exponentially large set of tours over which we can optimize efficiently, and they lead to well-solvable special cases of the tsp. Combined with a procedure for generating suitable permutations of the vertices, heuristic solutions to tsp can be obtained by computing optimal pyramidal tours with respect to the generated orders [12].

We will show that the classic $\mathcal{O}(n^2)$ dynamic program for finding bitonic tours in the Euclidean plane is far from optimal: by an appropriate use of dynamic geometric data structures, the running time can be reduced to $\mathcal{O}(n \log^2 n)$. To the best of our knowledge, this presents the first improvement in finding bitonic tours since the problem was popularized in *Introduction to Algorithms* [15] in 1991. In fact, we prove the stronger result that an optimal *pyramidal* tour among n points in the plane can be computed in $\mathcal{O}(n \log^2 n)$ time with respect to any given linear order on the points. Our techniques extend to the related Bottleneck Pyramidal tsp problem in the plane, where the goal is to find a pyramidal tour among the cities that minimizes the length of the longest edge. We prove that the underlying decision problem (given a linearly ordered set of points and a bottleneck value B, is there a pyramidal tour of the points whose longest edge has length at most B?) can be solved in $\mathcal{O}(n \log n)$ time, while the underlying optimization version (given a linearly ordered set of points, compute a bitonic tour that minimizes the length of the longest edge) can be solved in $\mathcal{O}(n \log^3 n)$ time. For the decision version of the bottleneck problem, we prove a matching $\Omega(n \log n)$ time lower bound in the algebraic computation tree model by a reduction from Set Disjointness with integer inputs [34]; this reduction even applies to the bitonic setting where the points are ordered from left to right.

***k*-OPT in the graph setting.** The complexity of k-OPT has been analyzed using the framework of parameterized complexity theory. Marx [28] proved that deciding whether there is a k-move that improves a given tour is W[1]-hard parameterized by k, giving evidence that there is no algorithm with runtime $f(k) \cdot n^{\mathcal{O}(1)}$. Guo *et al.* [22] refined this result and proved that, under the *Exponential Time Hypothesis* [23], there is no algorithm that determines whether a tour in a weighted complete graph can be improved by a k-move in time $f(k) \cdot n^{o(k/\log k)}$ for any function f. This lower bound shows that the exponent of n in the runtime of any k-OPT algorithm must grow almost linearly with k. The next question that we settle in this paper is: can one do better than $\mathcal{O}(n^k)$ for finding a k-OPT improvement? The answer turns out to depend on the value of k. For 2-OPT, an easy adversarial argument shows that any deterministic algorithm must inspect all the edge weights. This gives a trivial lower bound of $\Omega(n^2)$, matching the upper bound. For larger values of k, the question becomes more interesting.

The 3-OPT DETECTION problem asks whether the weight of a given tour can be reduced by some 3-move. We show that it is unlikely that 3-OPT DETECTION with weights in the range $[-M, \ldots, M]$ allows an algorithm with a *truly subcubic* runtime of $\mathcal{O}(n^{3-\varepsilon} \operatorname{polylog}(M))$ for $\varepsilon > 0$. We prove that the NEGATIVE EDGE-WEIGHTED TRIANGLE problem (given an edge-weighted graph, is there a triangle of negative weight?) reduces to 3-OPT DETECTION by a reduction that takes $\mathcal{O}(n^2)$ time and increases the size of the graph by only a constant factor. As NEGATIVE EDGE-WEIGHTED TRIANGLE is equivalent to ALL-PAIRS SHORTEST PATHS in weighted digraphs (APSP) with respect to having truly subcubic algorithms [33], a truly subcubic algorithm for 3-OPT DETECTION would contradict the APSP conjecture [2, 3] which states that APSP cannot be solved in truly subcubic time. We also give a reduction in the other direction: finding a 3-OPT improvement reduces to finding a negative edge-weighted triangle. Consequently, 3-OPT DETECTION is *equivalent* to NEGATIVE EDGE-WEIGHTED TRIANGLE and APSP with respect to truly subcubic runtimes. This adds yet another classic problem to the growing list of such equivalent problems [2, 33].

As a final result in this direction, we design an algorithm that finds the best k-OPT improvement in weighted n-vertex complete graphs in $\mathcal{O}(n^{\lfloor 2k/3 \rfloor + 1})$ time for each fixed value of k. For $k = 2$ and $k = 3$, this expression simply boils down to the straightforward time complexities of $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ for 2-OPT and 3-OPT respectively. For $k \geq 4$, however, our result yields a substantial improvement over the trivial $\mathcal{O}(n^k)$ time bound. For example, 4-OPT can be solved in $\Theta(n^3)$ time, matching the best-known algorithm for 3-OPT. The algorithm mixes enumeration of partial solutions with a simple dynamic program.

Faster 2-OPT in the repeated setting and in the planar setting. For the 2-OPT problem in graphs, the runtime for finding a single tour improvement cannot be improved below the trivial $\Theta(n^2)$. However, in the context of local search we are often interested in *repeatedly* finding tour improvements. It is therefore natural to consider whether speedups can be obtained when repeatedly finding improving tours on the same TSP instance. We prove that this is indeed the case: after $\mathcal{O}(n^2)$ preprocessing time, one can repeatedly find the best 2-OPT improvement in $\mathcal{O}(n \log n)$ time per iteration.

The quadratic lower bound for 2-OPT applies only in the graph setting. This raises the question: can we solve 2-OPT faster for points in the plane? We show the answer is yes, by giving an algorithm for 2-OPT DETECTION with runtime $\mathcal{O}(n^{8/5+\varepsilon})$ for all $\varepsilon > 0$. Similarly, 3-OPT DETECTION can be solved in expected time $\mathcal{O}(n^{80/31+\varepsilon})$.

2 Faster pyramidal TSP

In this section we show that the pyramidal TSP and the bottleneck pyramidal TSP problem in the plane can be solved in subquadratic time. For simplicity we only show how to compute the value of an optimal solution; the actual tour can be computed in the standard manner.

Let P be the ordered input set of n points with distinct x-coordinates in the plane. Our algorithm will consider the points in P in order, and maintain a collection of partial solutions that are locally optimal. To make this precise, define $P_i := \{p_1, \dots, p_i\}$ to be the first i points in P. A *partial solution* for P_i, for some $1 \le i \le n$, is a pair P', P'' of monotone paths (w.r.t. the order on P) that together visit all the points in P_i and that only share p_1. We call a partial solution for P_i an (i,j)*-partial tour*, for some $1 \le j < i$, if one of the paths ends at p_i – this is necessarily the case in a partial solution for P_i – and the other path ends at p_j.

Our starting point is the standard dynamic-programming solution. It uses a 2-dimensional table[1] $A[1..n, 1..n]$, where $A[i,j]$, for $1 \le j < i \le n$, is defined as the minimum length of an (i,j)-partial tour; for $i \le j \le n$ the entries $A[i,j]$ are undefined. We can compute the entries in the table row by row, using the recursive formula

$$A[i+1,j] \;=\; \begin{cases} A[i,j] + |p_i p_{i+1}| & \text{if } 1 \le j < i \\ \min_{1 \le k < i} \left(A[i,k] + |p_k p_{i+1}|\right) & \text{if } j = i \end{cases} \tag{1}$$

where $A[2,1] = |p_1 p_2|$. Let us briefly verify this recurrence. For $(i+1,j)$-partial tours with $j < i$, the path P' that visits p_{i+1} must also visit p_i: the other path P'' ends at index $j < i$ and the monotonicity requirement ensures P'' cannot visit i and go back to j. So for $j < i$ any $(i+1,j)$-partial tour consists of an (i,j)-partial tour together with the segment $p_i p_{i+1}$. For $(i+1,i)$-partial tours, the predecessor of p_{i+1} cannot be p_i, since a path ends at p_i. Hence an $(i+1,i)$-partial tour consists of an (i,k)-partial tour for some $1 \le k < i$ together with the segment $p_k p_{i+1}$. The cheapest combination yields the best partial tour.

After computing the last row of A, the minimum length of a pyramidal tour can be found by computing $\min_{1 \le k < n} \left(A[n,k] + |p_k p_n|\right)$. There are $\mathcal{O}(n^2)$ entries in A of the first type that each take constant time to evaluate. There are $\mathcal{O}(n)$ entries of the second type that need time $\Theta(n)$. Hence the dynamic program can be evaluated in $\mathcal{O}(n^2)$ time.

Our subquadratic algorithm is based on the following two observations. First, any two subsequent rows $A[i, 1..n]$ and $A[i+1, 1..n]$ are quite similar: the entries $A[i+1,j]$, for $j < i$, can all be obtained from $A[i,j]$ by adding the same value, namely $|p_i p_{i+1}|$. Second, the computation of $A[i+1,i]$ can be sped up using appropriate geometric data structures. Thus our algorithm will maintain a data structure that implicitly represents the current row and allows for fast queries and so-called bulk updates (see below).

Recall that $P_i := \{p_1, \dots, p_i\}$. The point that defines $\min_{1 \le k < i} \left(A[i,k] + |p_k p_{i+1}|\right)$ is the point $p_k \in P_{i-1}$ closest to the query point $q := p_{i+1}$ if we use the additively weighted distance function

$$\mathrm{dist}(p_k, q) := w_k + |p_k q|, \tag{2}$$

where $w_k := A[i,k]$ is the weight of p_k. Thus we need a data structure for storing a weighted point set that supports the following operations:

[1] Some of our results can also be obtained from an alternative DP with n states. As we need the 2-dimensional approach for Theorem 4, we present all our results in this setting.

- perform a *nearest-neighbor query* with a query point q, which reports the point p_k closest to q according to the additively weighted distance function,
- perform a *bulk update* of the weights, which adds a given value Δ to the weights of all the points currently stored in the data structure;
- *insert* a new point with a given weight into the data structure.

Answering nearest-neighbor queries for the weighted point set P can be done by performing point location in the *additively weighted Voronoi diagram* [18] of P augmented by a point location data structure [32]. This (static) data structure has size $\mathcal{O}(n)$, can be computed in $\mathcal{O}(n \log n)$ time, and allows for $\mathcal{O}(\log n)$-time queries. To allow for insertions we use the logarithmic method [8]. The logarithmic method makes a data structure semi-dynamic by storing $\mathcal{O}(\log n)$ static data structures of increasing size (resulting in an additional log-factor in the query time). The main observation is that we can handle bulk updates by storing a correction term for the weights with each of the static additively weighted Voronoi diagrams. The additively-weighted nearest neighbor structure does not change when adding the same constant to each point weight, which means we do not have to update the Voronoi diagrams when performing bulk updates. This leads to an implementation that supports each operation in $\mathcal{O}(\log^2 n)$ amortized time. The details are deferred to the full version. Using the data structure we obtain the following theorem.

▶ **Theorem 1.** *Let P be an ordered set of n points in the plane. Then we can compute a minimum-length pyramidal tour for P in $\mathcal{O}(n \log^2 n)$ time and using $\mathcal{O}(n)$ storage.*

Proof. We aim to speed up the classic dynamic-programming algorithm using the data structure described above. Instead of computing the entire dynamic programming table A explicitly, we maintain an implicit representation of one row of the table and compute the rows one by one. The i-th row of A has $i-1$ well-defined entries. We define an implicit representation of row i to be an instance of the data structure storing the weighted point set $P_{i-1} = \{p_1, \ldots, p_{i-1}\}$ such that $w(p_j) = A[i, j]$. The first nontrivial row in A is the second row, $A[2, 1..n]$. An implicit representation for that row consists of the point p_1 of weight $A[2, 1] = |p_1 p_2|$.

If we have an implicit representation of row i, we can efficiently obtain an implicit representation of row $i+1$, as we describe next. By our choice of implicit representation, the value of $A[i+1, i]$ according to (1) is exactly the distance from p_{i+1} to its closest neighbor in the data structure under the additively weighted distance function. Hence, the value of k that minimizes the lower expression in (1) can be found by a nearest neighbor query with p_{i+1}. We can therefore transform a representation of row i into a representation for row $i+1$ as follows:

1. Query with point p_{i+1} to find the value $A[i+1, i]$ and remember this value.
2. Perform a bulk update to increase the weight of the points $p_1, \ldots, p_{i-1}$ that are already in the structure by $\Delta := |p_i p_{i+1}|$. Recall that for cells j with $1 \le j < i$ their value in row $i+1$ is obtained from their value in row i by adding $|p_i p_{i+1}|$.
3. Insert point p_i of weight $A[i+1, i]$ into the structure.[2]

It is easy to verify that this yields an implicit representation of row $i+1$. Since a representation of the first nontrivial row can be found in constant time, and each successive row can be computed from the previous using three data structure operations that take $\mathcal{O}(\log^2 n)$

[2] We could also insert p_i with weight $A[i+1, i] - \Delta$. This way we would not have to subtract Δ from the weights of $p_1, \ldots, p_{i-1}$ in Step 2, and the bulk updates are not needed. As they are trivial in our data structure, we prefer the version that keeps the correspondence between weights and $A[i, j]$ values.

amortized time each, it follows that an implicit representation of the final row can be computed in $\mathcal{O}(n\log^2 n)$ time. The minimum cost of a pyramidal tour is $\min_{1\leq k<n}(A[n,k]+|p_kp_n|)$, which can be found by querying the representation of the final row with point p_n. ◀

Bottleneck pyramidal TSP. Using a similar global approach but different supporting data structures we can also solve the bottleneck version of the problem – here the goal is to minimize the length of the longest edge in the tour – in subquadratic time. For the decision version of the problem we need the following result.

▶ **Theorem 2.** *We can maintain a collection $\mathcal{D}$ of n congruent disks in a data structure such that we can decide in $\mathcal{O}(\log n)$ time if a query point q lies in* $\mathrm{Union}(\mathcal{D})$. *The data structure uses $\mathcal{O}(n)$ storage and a new disk can be inserted into $\mathcal{D}$ in $\mathcal{O}(\log n)$ amortized time.*

This result is obtained as follows. Assume the disks have radius $\sqrt{2}$ and consider the integer grid. Let $\mathcal{D}(C)\subseteq\mathcal{D}$ be the set of disks whose centers lie inside a grid cell C. To decide if $q\in\mathrm{Union}(\mathcal{D})$ we need to test if $q\in\mathrm{Union}(\mathcal{D}(C))$ for $O(1)$ grid cells C that are sufficiently close to q. Now consider a cell C with $\mathcal{D}(C)\neq\emptyset$. Obviously C itself is completely covered by $\mathrm{Union}(\mathcal{D}(C))$. Let $\ell_{\mathrm{top}}(C)$ be the line containing the top edge of C. Then the part of $\mathrm{Union}(\mathcal{D}(C))$ above $\ell_{\mathrm{top}}(C)$ – the other parts are handled similarly – is x-monotone. Moreover, we can show that each disk $D_i\in\mathcal{D}(C)$ contributes at most one arc to the boundary of $\mathrm{Union}(\mathcal{D}(C))$ above $\ell_{\mathrm{top}}(C)$, and the left-to-right order of the contributed arcs is consistent with the left-to-right order of the corresponding disk centers. Using this fact, we can do point locations and insertions in $O(\log n)$ time. Details can be found in the full version.

Combining the global technique of the previous section with Theorem 2 we obtain the following theorem.

▶ **Theorem 3.** *Let P be an ordered set of n points in the plane, and let $B>0$ be a given parameter. Then we can decide in $\mathcal{O}(n\log n)$ time and using $\mathcal{O}(n)$ storage if P admits a pyramidal tour whose longest edge has length at most B. This problem requires $\Omega(n\log n)$ time in the algebraic computation tree model of computation.*

The algorithm for the decision version does not easily extend to solve the minimization version of the problem. We therefore design a specialized data structure – a tree storing unions of disks and (regular) Voronoi diagrams – that allows us to obtain the following result.

▶ **Theorem 4.** *Let P be an ordered set of n points in the plane. Then we can compute a pyramidal tour whose bottleneck edge has minimum length in $O(n\log^3 n)$ time and using $\mathcal{O}(n\log n)$ storage.*

3 The k-OPT problem in general graphs

In this section we change the perspective from Euclidean problems to the TSP in general graphs. A *tour* of an undirected graph G is a Hamiltonian cycle in the graph. Depending on the context, we may treat a tour as a permutation of the vertex set or as a set of edges. We consider undirected, weighted complete graphs to model symmetric TSP inputs. The weight of a tour is simply the sum of the weights of its edges. Recall that a *k-move* of a tour T is an operation that replaces a set of k edges in T by another set of k edges from G in such a way that the result is a valid tour. In degenerate cases, such an operation may delete and reinsert the same edge. The associated decision problem is defined as follows.

> k-OPT DETECTION
> **Input:** A complete undirected graph G along with a (symmetric) distance function $d\colon E(G) \to \mathbb{N}$, an integer k, and a tour $T \subseteq E(G)$.
> **Question:** Is there a k-move that strictly improves the cost of T?

The optimization problem k-OPT OPTIMIZATION is to compute, given a tour in a graph, a k-move that gives the largest cost improvement, or report that no improving k-move exists.

3.1 On truly subcubic algorithms for 3-OPT

We say that an algorithm for n-vertex graphs with integer edge weights in the range $[-M, \ldots, M]$ runs in *truly subcubic time* if its runtime is bounded by $\mathcal{O}(n^{3-\varepsilon} \operatorname{polylog}(M))$ for some constant $\varepsilon > 0$. Vassilevska-Williams and Williams [33] introduced a framework for relating the truly subcubic solvability of several classic problems to each other. We use it to show that the existence of a truly subcubic algorithm for 3-OPT is unlikely. Their framework uses a notion of subcubic reducibility based on Turing reducibility [33, §IV] that solves one instance of problem A by repeatedly solving inputs of problem B. For our applications, simple reductions suffice that transform one input of problem A into one input of problem B of roughly the same size, in $\mathcal{O}(n^2)$ time.[3] Such reductions preserve the existence of truly subcubic algorithms, so we take this simpler viewpoint. The following problem is the starting point for our reductions.

> NEGATIVE EDGE-WEIGHTED TRIANGLE
> **Input:** An undirected, complete graph G and a weight function $w\colon E(G) \to \mathbb{Z}$.
> **Question:** Does G contain a triangle whose total edge-weight is negative?

Vassilevska-Williams and Williams [33, Thm. 1.1] proved that NEGATIVE EDGE-WEIGHTED TRIANGLE has a truly subcubic algorithm if and only if the ALL-PAIRS SHORTEST PATHS problem on digraphs with non-negative integral edge weights has a truly subcubic algorithm.

▶ **Lemma 5.** NEGATIVE EDGE-WEIGHTED TRIANGLE *can be reduced to* 3-OPT DETECTION *in time $\mathcal{O}(n^2)$, increasing the size of the graph and the largest weight by a constant factor.*

Proof. Consider an instance (G, w) of NEGATIVE EDGE-WEIGHTED TRIANGLE, and let $v_1, \ldots, v_n$ be an enumeration of the vertices of G. Let M be the largest absolute value of an edge weight. We introduce an instance of 3-OPT DETECTION that consists of $2n$ vertices $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$, where the starting tour T uses the ordering $a_1, b_1, a_2, b_2, \ldots, a_n, b_n$. The (symmetric) distances $d(\cdot, \cdot)$ between these vertices are defined as follows:

- $d(a_i, b_i) = 0$ for $1 \leq i \leq n$;
- $d(b_n, a_1) = -3M$, and $d(b_i, a_{i+1}) = -3M$ for $1 \leq i \leq n-1$;
- $d(a_i, b_j) = w(\{v_i, v_j\})$ for $1 \leq i < j \leq n$;
- $d(b_i, a_j) = w(\{v_i, v_j\})$ for $1 \leq i < j-1 \leq n-1$;
- $d(a_i, a_j) = d(b_i, b_j) = 3M$ for $1 \leq i \neq j \leq n$.

(For convenience, we allow distances to be negative in this construction. One easily moves to non-negative distances by adding the constant $4M$ to all distances.)

▶ **Claim 6.** *The constructed instance of* 3-OPT DETECTION *allows an improving* 3-OPT *move, if and only if the graph G contains a triangle of negative edge-weight.*

[3] We assume that simple arithmetic on weights can be done in constant time. The $\operatorname{polylog}(M)$ factors used in the framework originate from repeated executions to perform binary search on weight values.

Proof. ($\Leftarrow$) Assume that the vertices v_i, v_j, v_k span a triangle of negative edge-weight in G for $i < j < k$. We remove the three edges $\{a_i, b_i\}$, $\{a_j, b_j\}$, and $\{a_k, b_k\}$ from tour T, and we reconnect the resulting pieces by the three edges $\{a_i, b_j\}$, $\{a_j, b_k\}$, and $\{a_k, b_i\}$. The three removed edges have total length 0, while the three inserted edges have negative total length.

($\Rightarrow$) Now assume that there exists an improving 3-move for tour T. This improving move cannot remove any edge $\{b_i, a_{i+1}\}$ or $\{b_n, a_1\}$, as these edges have length $-3M$ while all newly inserted edges have non-negative length. Consequently, the three removed edges will be $\{a_i, b_i\}$, $\{a_j, b_j\}$, and $\{a_k, b_k\}$ for some $i < j < k$. As these three edges have total length 0, the total length of the three inserted edges must be strictly negative. The edges $\{a_x, a_y\}$ and $\{b_x, b_y\}$ all have length $3M$, while the edges $\{a_x, b_y\}$ all have length between $-M$ and M. This implies that every inserted edge is either of the type $\{a_x, b_y\}$, or coincides with one of the removed edges. Suppose for the sake of contradiction that one of the inserted edges coincides with a removed edge $\{a_k, b_k\}$, so that we are actually dealing with a 2-move. Then the two inserted edges in the 2-move must be $\{a_i, a_j\}$ and $\{b_i, b_j\}$, so that the new tour is by $6M$ *longer* than the old tour T. This contradiction leaves only two possibilities for the three inserted edges: either $\{a_i, b_j\}$, $\{a_j, b_k\}$, $\{a_k, b_i\}$, or $\{a_i, b_k\}$, $\{a_k, b_j\}$, $\{a_j, b_i\}$ (of which the latter is actually not a valid 3-move). Since the total length of the three inserted edges is strictly negative, the three vertices v_i, v_j, v_k form a triangle of strictly negative weight in G. ◀

The claim shows the correctness of the reduction. It is easy to perform in $\mathcal{O}(n^2)$ time. ◀

There is an analogous reduction in the other direction, which can be found in the full version. Together, these lemmata show the equivalence of finding negative-weight triangles and detecting improving 3-OPT moves. From our reductions and the results of Vassilevska-Williams and Williams [33, Thm. 1.1], we obtain the following theorem.

▶ **Theorem 7.** *There is a truly subcubic algorithm for* 3-OPT DETECTION *if and only if there is such an algorithm for* ALL-PAIRS SHORTEST PATHS *on weighted digraphs.*

3.2 A fast k-OPT algorithm

We will prove that the k-OPT OPTIMIZATION problem can be solved significantly faster than $\Theta(n^k)$ when $k \geq 4$. To this end, we first analyze the structure of k-OPT moves. Consider a k-move for a given tour $T \subseteq E(G)$, and let $e_1, \ldots, e_k$ be the removed edges with $e_i = \{v_{2i-1}, v_{2i}\}$. We assume throughout that these vertices (and edges) are indexed in such a way that T traverses the vertices v_i in order of increasing index. We assume furthermore that the vertices $v_1, \ldots, v_{2n}$ are pairwise distinct; all our arguments also go through without this assumption, but the notation becomes more complicated in the equality case. The k edges that are then inserted into T are denoted $f_1, \ldots, f_k$. The *signature* of this k-move is a permutation π of $\{1, \ldots, 2k\}$, such that v_j and $v_{\pi(j)}$ form the endpoints of one of the edges $f_1, \ldots, f_k$; see Fig. 1.

Note that the removed edges $e_1, \ldots, e_k$ together with the signature π fully determine the k-move (and in particular determine the inserted edges $f_1, \ldots, f_k$).

Note furthermore that not every permutation π yields a feasible signature that corresponds to some k-move: First, in a feasible signature $\pi(i) = j$ always implies $\pi(j) = i$, and we will always have $\pi(i) \neq i$. Secondly, in a feasible signature the edge set that results from T by removing $e_1, \ldots, e_k$ and by inserting $f_1, \ldots, f_k$ must form a single Hamiltonian cycle – it must never form a collection of two or more cycles. It is easy to check whether a given permutation π constitutes a feasible signature, and to enumerate all feasible signatures.

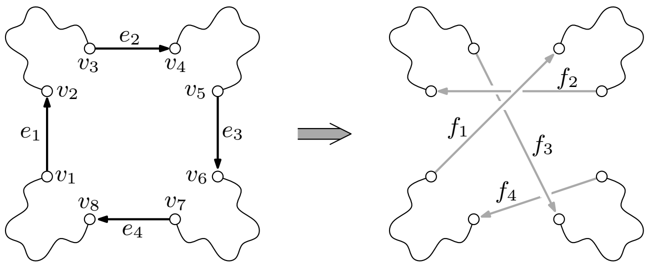

Figure 1 A 4-change with signature 4,5,7,1,2,8,3,6. Edges e_1 and e_4 are non-interfering. As we work on symmetric TSP, the graph and distance function are undirected; the arc directions merely indicate the traversal direction with respect to an arbitrary orientation of the tour.

We say that two of the removed edges e_i and e_j *interfere* with each other in a k-move, if there exists an inserted edge f that connects one of the endpoints of e_i to an endpoint of e_j.

▶ **Lemma 8.** *For any signature π, we can find a subset $E_\pi \subseteq \{e_1, \ldots, e_k\}$ of at least $\lceil k/3 \rceil$ removed edges that are pairwise non-interfering.*

Proof. The $2k$ edges $e_1, \ldots, e_k$ and $f_1, \ldots, f_k$ induce a set of cycles on the vertices $v_1, \ldots, v_{2k}$. If such a cycle contains an even number of removed edges, say 2ℓ, we put every other removed edge along this cycle into E_π; this yields ℓ out of 2ℓ edges for E_π. If the cycle contains only a single removed edge, we put this single edge into E_π; this yields one out of one edge for E_π. If the cycle contains an odd number of removed edges, say $2\ell + 1 \geq 3$, we ignore the first removed edge and then put every other removed edge along the cycle into E_π; this yields ℓ out of $2\ell + 1$ edges for E_π. The weakest contribution to E_π comes from cycles with three removed edges, which yield only one out of three edges for E_π. The claimed bound $\lceil k/3 \rceil$ follows. ◀

▶ **Theorem 9.** *For every fixed $k \geq 3$, the* k-OPT OPTIMIZATION *problem on an n-vertex graph can be solved in $\mathcal{O}(n^{\lfloor 2k/3 \rfloor + 1})$ time.*

Proof. For computing the best k-move for tour T, it is sufficient to compute for every feasible signature π – for fixed k there are only $\mathcal{O}(1)$ such signatures – the best k-move for tour T with that particular signature. This is done as follows. We first determine a set E_π of pairwise non-interfering edges according to the above lemma. Then we enumerate and handle all possible cases for the locations of the $\lfloor 2k/3 \rfloor$ removed edges not in E_π along T. This yields $\mathcal{O}(n^{\lfloor 2k/3 \rfloor})$ cases to handle, and every such case will be handled in $\mathcal{O}(n)$ time; note that this yields the claimed complexity. In handling a case, the positions of the removed edges not in E_π are frozen, while the edges in E_π have to be embedded into T. The cost of a k-move with signature π decomposes into two parts:

- The first part consists of the total weight of all frozen edges (which is subtracted) and the total weight of inserted edges between frozen edges (which is added).
- The second part consists of the individual contributions of the edges in E_π. For an edge $e \in E_\pi$ and an edge $e' \in T$, the cost of embedding e into e' equals the weight of the two inserted edges adjacent to e minus the weight of e'. As the edges in E_π are pairwise non-interfering, their individual cost contributions do not interact with each other.

As the cost of the first part is fixed in every considered case, our goal is to minimize the total cost of the second part. The frozen edges subdivide the tour T into a number of tour pieces, and we have to find the cheapest way of embedding the corresponding edges from E_π

into such a tour piece. The following paragraph sketches a straightforward dynamic program for finding the optimal embedding for each tour piece in time proportional to the length of the piece. As the length of all tour pieces combined is $\mathcal{O}(n)$, every case is indeed handled in time $\mathcal{O}(n)$.

We are essentially dealing with the following optimization problem. There are r locations $L_1, \dots, L_r$ (the edges along tour T between two consecutive frozen edges) and s objects $O_1, \dots, O_s$ (the edges in E_π that should be embedded between the two considered frozen edges). The objects are to be embedded into the locations, so that the location of object O_i always precedes the location of object O_{i+1}. The cost of embedding object O_i into location L_j is denoted $c(i, j)$. For $1 \leq x \leq s$ and $1 \leq y \leq r$, let $V(x, y)$ denote the smallest possible cost incurred by embedding the first x objects $O_1, \dots, O_x$ into the first y locations $L_1, \dots, L_y$. As $V(x, y)$ equals the minimum of $V(x, y-1)$ and $V(x-1, y-1) + c(x, y)$, all these values $V(x, y)$ can easily be computed in $\mathcal{O}(rs)$ time. In our situation, r is the length of the considered tour piece and $s \leq k$ is a constant that does not depend on the input; hence the complexity is indeed proportional to the length of the considered tour piece. ◀

4 Faster 2-OPT

In this section we show that it is possible to beat the quadratic barrier for 2-OPT in two important settings, namely when we want to apply 2-moves repeatedly, and in the Euclidean setting in the plane.

Repeated 2-OPT. In the repeated 2-OPT problem, we apply 2-OPT repeatedly (e.g. until no further improvements are possible). One can considerably speed up the 2-OPT computations at each of the iterations, except the first one. The following theorem gives our improvement for the 2-OPT OPTIMIZATION problem, where the goal is to find the best 2-move (rather than any 2-move that improves the tour).

▶ **Theorem 10.** *After $\mathcal{O}(n^2)$ preprocessing and using $\mathcal{O}(n^2)$ storage we can repeatedly solve the* 2-OPT OPTIMIZATION *problem in $\mathcal{O}(n \log n)$ time per iteration.*

The speedup claimed in the theorem relies on a tour representation that supports efficient 2-moves. To apply a 2-move that removes two edges e and e' and replaces them by the appropriate diagonal connections, one effectively has to reverse the part of the tour between e and e', or the part between e' and e. It can therefore take $\Omega(n)$ time to apply a 2-move to a tour represented as a sequence of vertices in an array. Chrobak *et al.* [14] give a speedup by storing the cities on the tour in an ordered balanced binary search tree. Each node in the tree stores a bit indicating whether the tour order is given by an in-order traversal of the subtree rooted there, or by the *reverse* of the in-order traversal. This allows a 2-move to be applied in $\mathcal{O}(\log n)$ time by manipulating reversal bits.

Our approach for repeated 2-OPT OPTIMIZATION is based on a similar data structure that represents tours in balanced search trees. However, instead of having only one tree that stores the current tour, we have n trees; one for each edge $e_1, \dots, e_n$ in the current tour. A query in the tree $\mathcal{T}(e_i)$ corresponding to edge e_i can be used to determine which edge e_j yields the most profitable 2-move together with e_i. After initializing these n trees, which takes $\mathcal{O}(n^2)$ time, an iteration of 2-OPT OPTIMIZATION can be performed as follows. For each e_i on the current tour, we query in tree $\mathcal{T}(e_i)$ to find the best 2-move that removes e_i and some unknown edge e_j in $\mathcal{O}(\log n)$ time. In this way we find the best overall 2-move which removes, say, edges e_i and e_j. We can update all trees $\mathcal{T}(e_\ell)$ for $\ell \neq i, j$ by deleting e_i

and e_j, and inserting the appropriate replacement edges. Using the reversal bits this can be done in $O(\log n)$ time. Trees $\mathcal{T}(e_i)$ and $\mathcal{T}(e_j)$ are destroyed; we build two new trees from scratch for the two new edges $e_{i'}$ and $e_{j'}$ that enter the tour. This gives $\mathcal{O}(n \log n)$ time per iteration.

It is likely that these techniques can be extended to speed up repeated 3-OPT as well. As the technical details become substantially more cumbersome, we do not pursue this direction.

The planar case. For points in the plane (and under the Euclidean metric) we can speed up 2-OPT computations by using suitable geometric data structures for semi-algebraic range searching; the details had to be omitted from this extended abstract. (Note that we do not consider the repeated version of the problem, but the single-shot version.) A similar approach can be used to speed up 3-OPT in the Euclidean setting in the plane. This leads to the following theorem.

▶ **Theorem 11.** *For any fixed $\varepsilon > 0$, 2-OPT DETECTION in the plane can be solved in $\mathcal{O}(n^{8/5+\varepsilon})$ time, and 3-OPT DETECTION in the plane can be solved in $\mathcal{O}(n^{80/31+\varepsilon})$ expected time.*

5 Conclusion

Revisiting the worst-case complexity of k-OPT and pyramidal TSP led to a number of new results on these classic problems. Some, such as the equivalence between 3-OPT and APSP with respect to having truly subcubic algorithms, rely on very recent work. Other results, such as the near-linear time algorithm for finding bitonic tours, and the k-OPT algorithm that beats the trivial $\mathcal{O}(n^k)$ upper bound, are obtained using classic techniques. In this respect, it is surprising that these results were not found earlier. These examples show that the availability of new lower bound machinery can inspire new algorithms.

Our findings suggest several directions for further research, both theoretical and applied. An interesting open problem regarding k-OPT DETECTION is whether the problem is fixed-parameter tractable when improving a given tour in an edge-weighted planar graph. This question was also asked by Marx [28] and Guo et al. [22]. Similarly, it is open whether the problem is fixed-parameter tractable when improving a given tour among points in the Euclidean plane. It would be interesting to settle the exact complexity of k-OPT in general weighted graphs. Is $\Theta(n^{\lfloor \frac{2k}{3} \rfloor + 1})$ the optimal running time for k-OPT DETECTION? When all weights lie in the range $[-M, \ldots, M]$, one can detect a negative triangle in an edge-weighted graph in time $\mathcal{O}(M \cdot n^{\omega})$ using fast matrix multiplication [4, 31, 35]. By our reduction, this gives an algorithm for 3-OPT DETECTION with weights $[-M, \ldots, M]$ in time $\mathcal{O}(M \cdot n^{\omega})$. Can similar speedups be obtained for k-OPT for larger k?

Given the great industrial interest in TSP, establishing the practical applicability of these theoretical results is an important follow-up step. Several of our results rely on data structures that are efficient in theory, but which are currently impractical. These include the additively-weighted Voronoi diagram used for pyramidal tours on points in the plane, and the semi-algebraic range searching data structures used to speed up 2-OPT DETECTION. In contrast, the $\mathcal{O}(n^{\lfloor 2k/3 \rfloor + 1})$ algorithm for finding the best k-move improvement is self-contained, easy to implement, and may have practical potential.

Acknowledgments. We are grateful to Hans L. Bodlaender, Karl Bringmann, and Jesper Nederlof for insightful discussions, an anonymous referee for the observation in Footnote 1, and Christian Knauer for the observation in Footnote 2.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proc. 56th FOCS*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.

2 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska-Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proc. 26th SODA*, pages 1681–1697, 2015. doi:10.1137/1.9781611973730.112.

3 Amir Abboud, Virginia Vassilevska-Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proc. 47th STOC*, pages 41–50, 2015. doi:10.1145/2746539.2746594.

4 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997. doi:10.1006/jcss.1997.1388.

5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proc. 47th STOC*, pages 51–58, 2015. doi:10.1145/2746539.2746612.

6 Md. Fazle Baki and Santosh N. Kabadi. Pyramidal traveling salesman problem. *Computers & OR*, 26(4):353–369, 1999. doi:10.1016/S0305-0548(98)00067-7.

7 Jon Louis Bentley. Experiments on traveling salesman heuristics. In *Proc. 1st SODA*, pages 91–99, 1990.

8 J.L. Bently and J.B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980. doi:10.1016/0196-6774(80)90015-2.

9 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *Proc. 55th FOCS*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

10 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Venkatesan Guruswami, editor, *Proc. 56th FOCS*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.

11 Rainer E. Burkard, Vladimir G. Deineko, René van Dal, Jack A. A. van der Veen, and Gerhard J. Woeginger. Well-solvable special cases of the traveling salesman problem: A survey. *SIAM Review*, 40(3):496–546, 1998. doi:10.1137/S0036144596297514.

12 J. Carlier and P. Villon. A new heuristic for the travelling salesman problem. *RAIRO – Operations Research*, 24:245–253, 1990.

13 Barun Chandra, Howard J. Karloff, and Craig A. Tovey. New results on the old k-OPT algorithm for the traveling salesman problem. *SIAM J. Comput.*, 28(6):1998–2029, 1999. doi:10.1137/S0097539793251244.

14 M. Chrobak, T. Szymacha, and A. Krawczyk. A data structure useful for finding hamiltonian cycles. *Theoretical Computer Science*, 71(3):419–424, 1990. doi:10.1016/0304-3975(90)90053-K.

15 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

16 G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6:791–812, 1958. doi:10.1287/opre.6.6.791.

17 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. *Algorithmica*, 68(1):190–264, 2014. doi:10.1007/s00453-013-9801-4.

18 Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987. doi:10.1007/BF01840357.

19 Michael L. Fredman, David S. Johnson, Lyle A. McGeoch, and G. Ostheimer. Data structures for traveling salesmen. *J. Algorithms*, 18(3):432–479, 1995. doi:10.1006/jagm.1995.1018.

20 P.C. Gilmore, E.L. Lawler, and D.B. Shmoys. Well-solved special cases. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*, pages 87–143. Wiley, New York, 1985.

21 Fred Glover. Finding a best traveling salesman 4-Opt move in the same time as a best 2-Opt move. *J. Heuristics*, 2(2):169–179, 1996. `doi:10.1007/BF00247211`.

22 Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondrej Suchý. The parameterized complexity of local search for TSP, more refined. *Algorithmica*, 67(1):89–110, 2013. `doi:10.1007/s00453-012-9685-8`.

23 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. `doi:10.1006/jcss.2001.1774`.

24 D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, Dordrecht, 2002.

25 D.S. Johnson and L.A McGeoch. The traveling salesman problem: A case study in local optimization. In E Aarts and J.K. Lenstra, editors, *Local search in combinatorial optimization*, pages 215–310. Wiley, Chichester, 1997.

26 Marvin Künnemann and Bodo Manthey. Towards understanding the smoothed approximation ratio of the 2-opt heuristic. In *Proc. 42nd ICALP*, pages 859–871, 2015. `doi:10.1007/978-3-662-47672-7_70`.

27 Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965. `doi:10.1002/j.1538-7305.1965.tb04146.x`.

28 Dániel Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008. `doi:10.1016/j.orl.2007.02.008`.

29 Ioannis Mavroidis, Ioannis Papaefstathiou, and Dionisios N. Pnevmatikatos. A fast FPGA-based 2-opt solver for small-scale euclidean traveling salesman problem. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 13–22, 2007. `doi:10.1109/FCCM.2007.40`.

30 Molly A. O'Neil and Martin Burtscher. Rethinking the parallelization of random-restart hill climbing: a case study in optimizing a 2-opt TSP solver for GPU execution. In *Proceedings of the 8th Workshop on General Purpose Processing using GPUs*, pages 99–108, 2015. `doi:10.1145/2716282.2716287`.

31 Liam Roditty and Virginia Vassilevska-Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *Proc. 52nd FOCS*, pages 180–189, 2011. `doi:10.1109/FOCS.2011.27`.

32 Jack Snoeyink. Point location. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry (2nd ed.)*. CRC Press, 2004.

33 Virginia Vassilevska-Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. 51th FOCS*, pages 645–654, 2010. `doi:10.1109/FOCS.2010.67`.

34 Andrew Chi-Chih Yao. Lower bounds for algebraic computation trees with integer inputs. *SIAM J. Comput.*, 20(4):655–668, 1991. `doi:10.1137/0220041`.

35 Gideon Yuval. An algorithm for finding all shortest paths using $n^{2.81}$ infinite-precision multiplications. *Inf. Process. Lett.*, 4(6):155–156, 1976. `doi:10.1016/0020-0190(76)90085-5`.