Fine-Grained Complexity Analysis of Two Classic TSP Variants

Based on the paper by Mark de Berg, Kevin Buchin, Bart M. P. Jansen, and Gerhard Woeginger in ICALP 2016

Philipp Schepper

August 23, 2018

Abstract

The paper shows a faster algorithm for bitonic TSP; for given points in the plane one asks for the shortest tour consisting of two monotone paths with respect to the left-right order of the points. While the previously best algorithm runs in $\mathcal{O}(n^2)$ time, the improved algorithm runs in time $\mathcal{O}(n \log^2 n)$. Also, we show a faster algorithm for k-Opt; here one wants to improve a given tour in a weighted graph by replacing k edges by k (different) edges. While the standard algorithm requires $\Theta(n^k)$ time, the improved algorithm runs in $\mathcal{O}(n^{\lfloor 2k/3 \rfloor + 1})$ time. Furthermore we outline the reduction from Negative Triangle to 3-Opt showing a conditional lower bound for the latter problem.

1 Introduction

Since the classical Traveling Salesman Problem (TSP) is **NP**-complete, it is unlikely to find polynomial-time algorithms for it. Therefore we focus on two variations of this problem.

Bitonic TSP The first problem is a restriction to special kinds of graphs and tours. For bitonic TSP one is given n points in the plane. These points are ordered with respect to their x-coordinates from left to right. The problem asks for a bitonic tour $v_1, v_{i_1}, \ldots, v_{i_r}, v_n, v_{j_1}, \ldots, v_{j_{n-r-2}}$ of minimal length. A bitonic tour is a Hamiltonian cycle whereby the x-coordinates increase on the subpath going from v_1 to v_n and decrease on the subpath from v_n to v_1.

Since the first version of *"Introduction to Algorithms"* by Cormen, Leierson, Rivest, and Stein from 1991, finding an $\mathcal{O}(n^2)$ time algorithm for bitonic TSP is an exercise for students. But until this paper, no faster algorithm was found. Therefore it is quite surprising that the algorithm can be improved by keeping the main idea the same and changing the data structure. This improved $\mathcal{O}(n \log^2 n)$ time algorithm is shown in the paper.

For simplicity we only consider bitonic TSP in this paper. But the original paper showed all the presented results for the more general pyramidal TSP in the plane. For this problem one is given an arbitrary ordering of points in the plane and searches for the shortest tour with respect to this ordering.

Furthermore, the problem of minimizing the maximum distance between two points in a tour (i.e. bottleneck pyramidal TSP in the plane) is also improved to a super-linear $\mathcal{O}(n \log^3 n)$ running time by a similar change of the data structures.

k-Opt The second variant of TSP follows a completely different approach. A lot of heuristics are known to produce "good" solutions for TSP instances in practice. Since these solutions are usually not optimal one can try to improve them further. For this we focus on k-moves. A k-move removes k edges from the given tour and replaces them by k edges such that they form a valid tour again. The length of the new tour must be smaller than the length of the old

tour. The k-Opt problem asks whether there exists such an improving k-move while the k-Opt Optimization problem searches for the k-move with largest improvement.

For the 3-Opt problem a reduction from and to Negative Triangle is shown, implying a conditional cubic lower bound for 3-Opt since it is conjectured that there is no subcubic algorithm for Negative Triangle. This implies the (conditional) optimality of the algorithm for 3-Opt.

Furthermore, the paper presents an improvement of the standard algorithm for k-Opt for $k > 3$. Whereas the standard algorithm is a brute-force approach trying all possible k-moves in the tour, the new algorithm exploits the structure of the k-moves. By this a significant runtime improvement from $\Theta(n^k)$ to $\mathcal{O}(n^{\lfloor 2k/3 \rfloor + 1})$ is shown. This implies 4-Opt Optimization is (currently) solvable as fast as 3-Opt Optimization.

Since the conditional lower bound does only hold for general graphs, the paper shows an improvement of the 2-Opt and 3-Opt problem for points in the plane. In this case, the problems can be solved in $\mathcal{O}(n^{8/5+\varepsilon})$ and $\mathcal{O}(n^{80/31+\varepsilon})$, respectively, for any fixed $\varepsilon > 0$. Additionally, repeated 2-Opt Optimization can be solved in $\mathcal{O}(n \log n)$ time per iteration after $\mathcal{O}(n^2)$ preprocessing.

In the following, we present the main ideas behind the major results of the paper. After showing a faster algorithm for bitonic TSP, we present the improved algorithm for k-Opt Optimization. As a last result we outline the reduction from Negative Triangle to 3-Opt.

2 A Faster Algorithm for bitonic TSP

As first result of the paper, a faster algorithm for bitonic TSP is shown. Suppose we are given n points in the plane, numbered accordingly to their x-coordinates from left to right. A bitonic tour is a Hamiltonian cycle starting at the left-most point and going to the right-most point by strictly increasing the x-coordinates, and then going back to the left-most point by strictly decreasing the x-coordinates. In the bitonic TSP problem, we ask for the shortest of these tours.

While the standard dynamic programming algorithm as shown in Figure 1a requires $\mathcal{O}(n^2)$ time, the improved algorithm needs only $\mathcal{O}(n \log^2 n)$ time. Therefore it is quite interesting that the main idea of the algorithm is still the same, the only change is in the used data structure. The vast speedup is obtained by the following observations including the use of additively weighted Voronoi diagrams (AWVD). The AWVD of m weighted points in the plane is a subdivision of the plane into at most m regions. A point q lies in the region of one of these given points p if p is the nearest neighbor of q while adding the weight of p to the distance.

- The results in the outer loop depend only on the results of the previous iteration.
- The values $A[i, 1 \dots i-1]$ are updated to $A[i+1, 1 \dots i-1]$ by adding $d(v_i, v_{i+1})$ to each entry.

$A[2,1] := d(v_1, v_2)$
For $i = 2, \dots, n-1$:
  For $j = 1, \dots, i-1$:
    $A[i+1, j] := A[i, j] + d(v_i, v_{i+1})$
  $A[i+1, i] = \min_{1 \le k < i}(A[i, k] + d(v_k, v_{i+1}))$

Return $\min_{1 \le k < n}(A[n, k] + d(v_k, v_n))$

(a) The old $\mathcal{O}(n^2)$ algorithm

Set up the data structure
For $i = 2, \dots, n-1$:
  Do a bulk update with weight $d(v_i, v_{i+1})$

  Perform a nearest-neighbor query for v_{i+1}
  Insert the new point into the data structure
Return $\min_{1 \le k < n}(\text{weight}(\text{point}_k) + d(v_k, v_n))$

(b) The new $\mathcal{O}(n \log^2 n)$ algorithm

Figure 1: The algorithms for bitonic TSP

- AWVDs allow bulk updates (change all weights by the same value).
- Entries of the matrix A can be treated as weights of the points in an AWVD.
- Nearest-neighbor query (minimum) is fast in these diagrams.
- Inserting points can be done fast.

With these observations one can easily speed up the algorithm to the one shown in Figure 1b. Since AWVDs allow to perform each operation (bulk updates, nearest-neighbor queries, and insertion of points) in $\mathcal{O}(\log^2 n)$ time, the new runtime follows immediately.

3 A faster algorithm for k-Opt

In the k-Opt Optimization problem, we are given a complete undirected graph G along with a distance function $d : E(G) \to \mathbb{N}$. Furthermore, we are given an initial tour $T \subseteq E(G)$. The output is a new valid tour with maximal improved cost, originating from the old tour T by replacing k edges by k (new) edges; this is called a (proper) k-move. The k-Opt problem is the decision variant of k-Opt Optimization, i.e. decide whether there is an improving k-move. While the classical algorithm runs in $\Theta(n^k)$ time by explicitly checking all possible k-moves, the paper presents an $\mathcal{O}(n^{\lfloor 2k/3 \rfloor + 1})$ time algorithm by exploiting the special structure of k-moves.

Formally, a k-move consists of deleted edges $e_1, \ldots, e_k$ with $e_i = (v_{2i-1}, v_{2i})$ and the inserted edges $f_1, \ldots, f_k$; notice that we only label those edges and vertices in the tour that are part of the k-move. We first describe how to compress a k-move. This is done by defining a permutation $\pi : [2k] \to [2k]$ such that v_j and $v_{\pi(j)}$ are connected by some $f_{j'}$ in the new tour. By this the deleted edges and the permutation fully determine the k-move. We call π the signature of the corresponding k-move.

The crucial point of the algorithm is the exploitation of non-interfering edges. Two removed edges e_i and e_j interfere with each other in a k-move if there is an edge f_l connecting two endpoints of these edges e_i and e_j. For each signature π we can find a subset $E_\pi \subseteq [k]$ of at least $\lceil k/3 \rceil$ indices of pairwise non-interfering edges. This follows since the deleted edges $e_1, \ldots, e_k$ and the inserted edges $f_1, \ldots, f_k$ induce a set of cycles; for each cycle we pick the index of every second deleted edge.

The fast algorithm repeats the following computation for each signature π (since k is fixed, there are only $\mathcal{O}(1)$ possible signatures): We first compute $\bar{E}_\pi = [k] \setminus E_\pi$. Although the removed edges are not known at the moment, it is possible to compute E_π because the set depends only on the relation of the edges to each other, i.e. the signature, and not on their actual position in the tour. By a brute-force approach we try out all possible $\mathcal{O}(n^{\lfloor 2k/3 \rfloor})$ choices of the deleted edges e_i with $i \in \bar{E}_\pi$. Since the remaining edges with indices in E_π can be chosen in an optimal way in $\mathcal{O}(n)$ time, the claimed runtime of $\mathcal{O}(n^{\lfloor 2k/3 \rfloor + 1})$ follows.

For this last step, we recall that the edges in E_π are non-interfering and can therefore be placed independently of each other, apart from the ordering constraints. This can be done by the following dynamic programming approach: Suppose we want to choose the first l edges in E_π from the first m unused edges of the tour. This is equal to the minimum of

- the cost of choosing the first $l-1$ edges in E_π from the first $m-1$ unused edges of the tour plus the cost of choosing the mth unused edge of the tour as the lth edge in E_π or
- the cost of choosing the first l edges of E_π from the first $m-1$ unused edges of the tour.

One can easily see there are $\mathcal{O}(k \cdot n)$ values to compute, yielding the claimed running time since k is assumed to be fixed.

4 Lower Bounds

Now we sketch the reduction from Negative Triangle to the decision problem 3-Opt. This yields a conditional cubic lower bound for 3-Opt (and 3-Opt Optimization) since it is conjectured that All-Pairs Shortest Paths (APSP) has no subcubic time algorithm and APSP and Negative Triangle are known to be equivalent.

For a given Negative Triangle instance with a complete graph $G = (V, E)$, nodes $V = \{v_1, \ldots, v_n\}$ and a weight function $w : E \to \mathbb{Z}$ we construct the 3-Opt instance as follows: Introduce for each node v_i two points a_i, b_i. Use the walk $a_1, b_1, a_2, \ldots, a_n, b_n, a_1$ as initial tour. Set M to be the maximum absolute weight of an edge in G. Then we define the following distances for the 3-Opt problem:

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

This instance of 3-Opt can clearly be constructed in $\mathcal{O}(n^2)$ time. For the correctness of the reduction, we first need to show the existence of an improving 3-move given a negative weight triangle v_i, v_j, v_k with $i < j < k$. This follows directly from choosing $(a_i, b_i), (a_j, b_j)$, and (a_k, b_k) as edges removed from the tour and (a_i, b_j), (a_k, b_i), and (a_j, b_k) as inserted edges for the new tour.

Second, for the other direction we assume the edges $(a_i, b_i), (a_j, b_j)$, and (a_k, b_k) are removed by the k-move for some $i < j < k$, otherwise we cannot improve the length of the tour. A simple case analysis of all possible reconnections yields, that the only way to form an improving cycle without reusing all removed edges is to insert the edges (a_i, b_j), (a_k, b_i), and (a_j, b_k) into the tour. By the definition of the 3-Opt instance, this corresponds to a triangle with negative weight and implies the correctness of the reduction.