Fine-Grained Complexity Analysis of Two Classic TSP Variants

Mark de Berg, Kevin Buchin,
Bart M. P. Jansen, and Gerhard Woeginger

43rd International Colloquium on Automata, Languages, and Programming
(ICALP 2016)

Philipp Schepper

June 12, 2018

Outline

Motivation

- Traveling Salesman Problem (TSP) is **NP**-hard
  $\Rightarrow$ Unlikely to have fast algorithms

Motivation

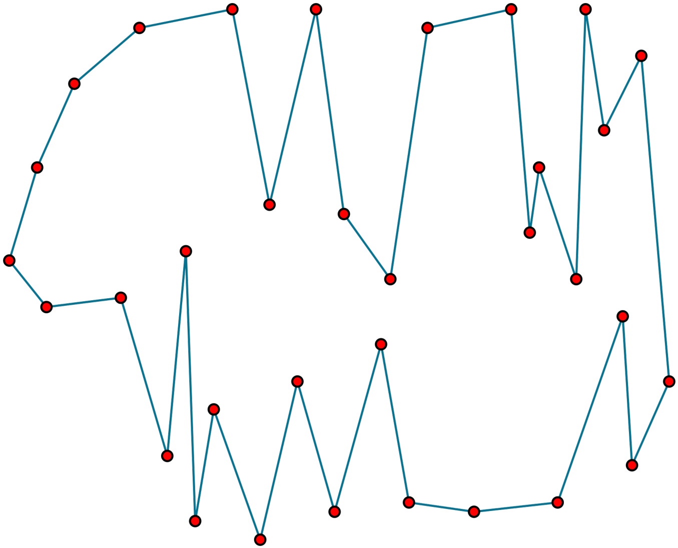
- Traveling Salesman Problem (TSP) is **NP**-hard
  $\Rightarrow$ Unlikely to have fast algorithms
- Solve relaxations instead or proof their hardness

Bitonic TSP

Definition: Bitonic TSP

Input: n nodes in the plane (with distinct x-coordinates)
Output: A shortest hamiltonian cycle consisting of two monotone path with respect to their left-right order.

Bitonic TSP



Bitonic TSP

Definition: Bitonic TSP

Input: n nodes in the plane (with distinct x-coordinates)
Output: A shortest hamiltonian cycle consisting of two monotone path with respect to their left-right order.

Theorem

Bitonic TSP can be solved in $\mathcal{O}(n^2)$ time.

Since 1991 finding this algorithm is an exercise for students!

Dynamic Program for Bitonic TSP

- Assume nodes are ordered s.t. the left-most node is v_1 and the right-most node is v_n
- $A \in \mathbb{R}^{n \times n}$
- $A[i,j] :=$ sum of the lengths of the *two* shortest disjoint bitonic tours both starting at v_1, ending at i and $j < i$, and covering all points $\{v_1, \ldots, v_i\}$.

1. $A[2,1] := d(v_1, v_2)$
2. For $i = 2, \ldots, n-1$:
3. For $j = 1, \ldots, i-1$:
4. $A[i+1, j] := A[i,j] + d(v_i, v_{i+1})$
5. $A[i+1, i] = \min_{1 \leq k < i}(A[i,k] + d(v_k, v_{i+1}))$
6. Return $\min_{1 \leq k < n}(A[n,k] + d(v_k, v_n))$

Observations

- New values depend only on results from previous step
- Most values are changed equally $(+d(v_i, v_{i+1}))$
- Search for minimum
- One new entry in table, i.e. $A[i+1, i]$

Observations

- New values depend only on results from previous step
- Most values are changed equally $(+d(v_i, v_{i+1}))$
- Search for minimum
- One new entry in table, i.e. $A[i+1, i]$

$\Rightarrow$ Speed up each operation!

Observations

- New values depend only on results from previous step
- Most values are changed equally $(+d(v_i, v_{i+1}))$
- Search for minimum
- One new entry in table, i.e. $A[i+1, i]$

$\Rightarrow$ Speed up each operation!

- Store only current values ($\rightsquigarrow$ fix i)
- Treat values as points with associated weight $A[i, k]$
- Perform bulk updates on weights $(+d(v_i, v_{i+1}))$
- Nearest-neighbor query
- Insert one new value

The required data structure

- Fast setup time and dynamic changeable
- Store weighted points in the plane
- Perform nearest-neighbor query
- Bulk updates (change weights of all nodes simultaneously)
- Insert point to data structure

Additively weighted Voronoi Diagrams

$q \in \mathcal{C}(p_i) :\Longleftrightarrow$
$\forall j \neq i : d(q, p_i) + w_i \leq d(q, p_j) + w_j$

Find $\arg\min_{1 \leq k < i}(A[i,k] + d(v_k, v_{i+1}))$
$\Rightarrow$ Find k s.t. $v_{i+1} \in \mathcal{C}(p_k)$.

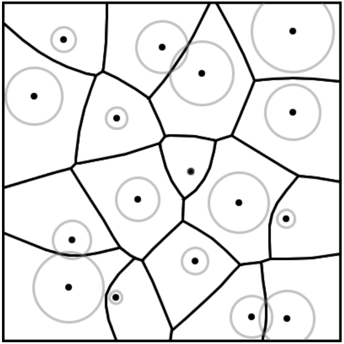


Image from `http://d.hatena.ne.jp/kaiseh/20091010/1255198573` .

Additively weighted Voronoi Diagrams

$q \in \mathcal{C}(p_i) :\Longleftrightarrow$
$\forall j \neq i : d(q, p_i) + w_i \leq d(q, p_j) + w_j$

Find $\arg\min_{1 \leq k < i}(A[i, k] + d(v_k, v_{i+1}))$
$\Rightarrow$ Find k s.t. $v_{i+1} \in \mathcal{C}(p_k)$.

- Setup: $\mathcal{O}(n \log n)$
- Nearest neighbor search: $\mathcal{O}(\log^2 n)$
- Bulk updates: $\mathcal{O}(\log n)$
- Insert point: $\mathcal{O}(\log^2 n)$

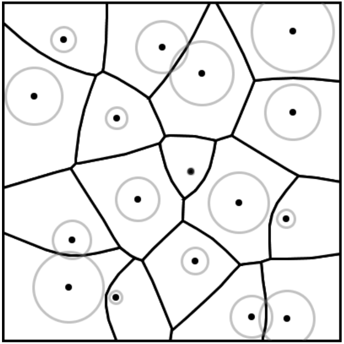


Image from http://d.hatena.ne.jp/kaiseh/20091010/1255198573 .

Faster bitonic TSP

Theorem 1

Bitonic TSP in the plane with n nodes can be solved in $\mathcal{O}(n \log^2 n)$ time.

Faster bitonic TSP

Theorem 1

Bitonic TSP in the plane with n nodes can be solved in $\mathcal{O}(n \log^2 n)$ time.

1. $A[2,1] := d(v_1, v_2)$
2. Set up the data structure
3. For $i = 2, \ldots, n-1$:
4. For $j = 1, \ldots, i-1$:
5. $A[i+1, j] := A[i, j] + d(v_i, v_{i+1})$
6. $A[i+1, i] = \min_{1 \leq k < i}(A[i, k] + d(v_k, v_{i+1}))$
7. Return $\min_{1 \leq k < n}(A[n, k] + d(v_k, v_n))$

Faster bitonic TSP

Theorem 1

Bitonic TSP in the plane with n nodes can be solved in $\mathcal{O}(n \log^2 n)$ time.

1. $A[2,1] := d(v_1, v_2)$
2. Set up the data structure
3. For $i = 2, \dots, n-1$:
4. For $j = 1, \dots, i-1$:
5. $A[i+1, j] := A[i, j] + d(v_i, v_{i+1})$
6. $A[i+1, i] = \min_{1 \leq k < i}(A[i,k] + d(v_k, v_{i+1}))$
7. Return $\min_{1 \leq k < n}(A[n,k] + d(v_k, v_n))$

Faster bitonic TSP

Theorem 1

Bitonic TSP in the plane with n nodes can be solved in $\mathcal{O}(n \log^2 n)$ time.

1. Set up the data structure
2. For $i = 2, \ldots, n-1$:
3. For $j = 1, \ldots, i-1$:
4. $A[i+1, j] := A[i, j] + d(v_i, v_{i+1})$
5. $A[i+1, i] = \min_{1 \leq k < i}(A[i, k] + d(v_k, v_{i+1}))$
6. Return $\min_{1 \leq k < n}(A[n, k] + d(v_k, v_n))$

Faster bitonic TSP

Theorem 1

Bitonic TSP in the plane with n nodes can be solved in $\mathcal{O}(n \log^2 n)$ time.

1. Set up the data structure
2. For $i = 2, \dots, n-1$:
3. For $j = 1, \dots, i-1$:
4. $A[i+1, j] := A[i, j] + d(v_i, v_{i+1})$
5. $A[i+1, i] = \min_{1 \leq k < i}(A[i, k] + d(v_k, v_{i+1}))$
6. Return $\min_{1 \leq k < n}(A[n, k] + d(v_k, v_n))$

Faster bitonic TSP

Theorem 1

Bitonic TSP in the plane with n nodes can be solved in $\mathcal{O}(n \log^2 n)$ time.

1. Set up the data structure
2. For $i = 2, \ldots, n-1$:
3.
4. Do a bulk update with weight $d(v_i, v_{i+1})$
5. $A[i+1, i] = \min_{1 \leq k < i}(A[i,k] + d(v_k, v_{i+1}))$
6. Return $\min_{1 \leq k < n}(A[n,k] + d(v_k, v_n))$

Faster bitonic TSP

Theorem 1

Bitonic TSP in the plane with n nodes can be solved in $\mathcal{O}(n \log^2 n)$ time.

1. Set up the data structure
2. For $i = 2, \ldots, n-1$:
3.
4. Do a bulk update with weight $d(v_i, v_{i+1})$
5. $A[i+1, i] = \min_{1 \leq k < i}(A[i,k] + d(v_k, v_{i+1}))$
6. Return $\min_{1 \leq k < n}(A[n,k] + d(v_k, v_n))$

Faster bitonic TSP

Theorem 1

Bitonic TSP in the plane with n nodes can be solved in $\mathcal{O}(n \log^2 n)$ time.

1. Set up the data structure
2. For $i = 2, \ldots, n-1$:
3.
4. Do a bulk update with weight $d(v_i, v_{i+1})$
5. Perform a nearest-neighbor query for v_{i+1}
6. Return $\min_{1 \leq k < n}(A[n, k] + d(v_k, v_n))$

Faster bitonic TSP

Theorem 1

Bitonic TSP in the plane with n nodes can be solved in $\mathcal{O}(n \log^2 n)$ time.

1. Set up the data structure
2. For $i = 2, \ldots, n-1$:
3.
4. Do a bulk update with weight $d(v_i, v_{i+1})$
5. Perform a nearest-neighbor query for v_{i+1}
6. Return $\min_{1 \leq k < n}(A[n, k] + d(v_k, v_n))$

Faster bitonic TSP

Theorem 1

Bitonic TSP in the plane with n nodes can be solved in $\mathcal{O}(n \log^2 n)$ time.

1. Set up the data structure
2. For $i = 2, \ldots, n-1$:
3.
4. Do a bulk update with weight $d(v_i, v_{i+1})$
5. Perform a nearest-neighbor query for v_{i+1}
6. Perform a nearest-neighbor query for v_n

k-OPT

TSP is **NP**-hard $\Rightarrow$ Unlikely to have a fast algorithm for exact solution

k-OPT

TSP is **NP**-hard $\Rightarrow$ Unlikely to have a fast algorithm for exact solution

Different approach:
Start with "good" solution and change it locally.

Usually:

- One start with solution found by some heuristic
- Improve solution in several rounds

The k-Opt Problems

Definition: (Proper) k-move

Replacement of k edges in a tour by k (different) edges such that the new tour is valid.

Definition: k-Opt

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

In k-Opt Optimization we ask for a k-move with largest cost improvement.

A first algorithm

Theorem

k-Opt Optimization can be solved in $\mathcal{O}(n^k)$ time for fixed k.

Proof

Label the vertices s.t. the tour is $v_1, \ldots, v_n, v_1$

1. For $i_1 = 1, \ldots, n-k+1$:
2. For $i_2 = i_1 + 1, \ldots, n-k+2$:
3. $\ldots$
4. For $i_k = i_{k-1} + 1, \ldots, n$:
5. Remove edges $\{v_{i_j}, v_{i_j+1}\}$ $\forall j$ from T.
6. Check for each combination of points whether they form a
7. feasible tour and improve the cost.

Assumption

Two removed/inserted edges do not share an endpoint!
Only to keep notation simpler.

A fast algorithm for k-Opt Optimization

Definition: Interfering edges

Two removed edges *interfere* with each other in a k-move if they are connected by an inserted edge.

A fast algorithm for k-Opt Optimization

Definition: Interfering edges

Two removed edges *interfere* with each other in a k-move if they are connected by an inserted edge.

Lemma 3.2

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.

A fast algorithm for k-Opt Optimization

Definition: Interfering edges

Two removed edges *interfere* with each other in a k-move if they are connected by an inserted edge.

Lemma 3.2

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.

Theorem 6

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.

A fast algorithm for k-Opt Optimization

Proof of Theorem 6

Graph G, $k \in SetN$

1. For all signatures π:
2. Compute E_π and $\bar{E}_\pi = \{a_1, \ldots, a_k\} \setminus E_\pi$
3. For all possible position of abstract edges $a_i \in \bar{E}_\pi$ in G:
4. Insert the edges between these edges ($\rightsquigarrow$ update cost)
5. Find optimal embedding of edges $a_i \in E_\pi$ into G

A fast algorithm for k-Opt Optimization

Proof of Theorem 6

Graph G, $k \in SetN$

1. For all signatures π:
2. Compute E_π and $\bar{E}_\pi = \{a_1, \ldots, a_k\} \setminus E_\pi$
3. For all possible position of abstract edges $a_i \in \bar{E}_\pi$ in G:
4. Insert the edges between these edges ($\rightsquigarrow$ update cost)
5. Find optimal embedding of edges $a_i \in E_\pi$ into G

Corollary

3-Opt Optimization and 4-Opt Optimization can be solved in $\mathcal{O}(n^3)$ time.

Lower Bounds

Lemma 3.1

Negative Triangle can be reduced to 3-Opt in time $\mathcal{O}(n^2)$ while increasing the size of the graph and the largest weight by a constant factor.

Lower Bounds

Lemma 3.1

Negative Triangle can be reduced to 3-Opt in time $\mathcal{O}(n^2)$ while increasing the size of the graph and the largest weight by a constant factor.

Corollary

The algorithms for 3-Opt Optimization and 4-Opt Optimization are optimal (assuming the APSP conjecture).

Proof of Lemma 3.1

- (G, w) Negative Triangle instance with symmetric w
- Define 3-Opt instance with initial tour $T = a_1, b_1, \ldots, a_n, b_n, a_1$:
  - $M := \max_{i,j \in [n]} |w(v_i, v_j)|$
  - $d(a_i, b_i) = 0 \quad \forall 1 \leq i \leq n$
  - $d(b_n, a_1) = d(b_i, a_{i+1}) = -3M \quad \forall 1 \leq i < n$
  - $d(a_i, b_j) = w(v_i, v_j) \quad \forall 1 \leq i < j \leq n-1$
  - $d(b_i, a_j) = w(v_i, v_j) \quad \forall 1 \leq i < j-1 \leq n-1$
  - $d(a_i, a_j) = d(b_i, b_j) = 3M \quad \forall i \neq j$

Reduction requires $\mathcal{O}(n^2)$ time.

Further results

Lemma C.1

3-Opt can be reduced to Negative Triangle in time $\mathcal{O}(n^2)$ while increasing the size of the graph and the largest weight by a constant factor.

Further results

Lemma C.1

3-Opt can be reduced to Negative Triangle in time $\mathcal{O}(n^2)$ while increasing the size of the graph and the largest weight by a constant factor.

Theorem 5

There is a truly subcubic algorithm for 3-Opt if and only if there is such an algorithm for APSP on weighted digraphs.

$\Rightarrow$ APSP equivalence for 3-Opt.

Further results

Theorem 4

Bottleneck pyramidal ($\approx$ bitonic) TSP with n ordered points in the plane can be solved in $\mathcal{O}(n \log^3 n)$ time.

Theorem 7

Repeated 2-Opt Optimization can be solved in $\mathcal{O}(n \log n)$ per iteration after $\mathcal{O}(n^2)$ preprocessing.

Theorem 8

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