richardwu.ca

# CS 466/666 Course Notes

Design and Analysis of Algorithms

Anna Lubiw • Fall 2018 • University of Waterloo

Last Revision: September 10, 2018

## Table of Contents

**1 September 10, 2018 1**
1.1 Overview . . . 1
1.2 Travelling Salesman Problem (TSP) . . . 1
1.3 Approach to NP-complete problems . . . 2
1.4 Metric TSP . . . 2

**Abstract**

These notes are intended as a resource for myself; past, present, or future students of this course, and anyone interested in the material. The goal is to provide an end-to-end resource that covers all material discussed in the course displayed in an organized manner. These notes are my interpretation and transcription of the content covered in lectures. The instructor has not verified or confirmed the accuracy of these notes, and any discrepancies, misunderstandings, typos, etc. as these notes relate to course's content is not the responsibility of the instructor. If you spot any errors or would like to contribute, please contact me directly.

# 1 September 10, 2018

## 1.1 Overview

**How to design algorithms** Assume: greedy, divide-and-conquer, dynamic programming

New: randomization, approximation, online algorithms

For one's basic repertoire, assume knowledge of basic data structures, graph algorithms, string algorithms.

**Analyzing algorithms** Assume: big Oh, worst case asymptotic analysis

New: amortized analysis, probabilistic analysis, analysis of approximation factors

**Lower Bounds** Assume: NP-completeness

New: hardness of approximation

## 1.2 Travelling Salesman Problem (TSP)

Given graph $(V, E)$ with weights on edges $W : E \to \mathbb{R}^{\geq 0}$ find a *TSP tour* (i.e. a cycle that visits every vertex exactly once and has minimum weight or min $\sum_{e \in C} w(e)$).

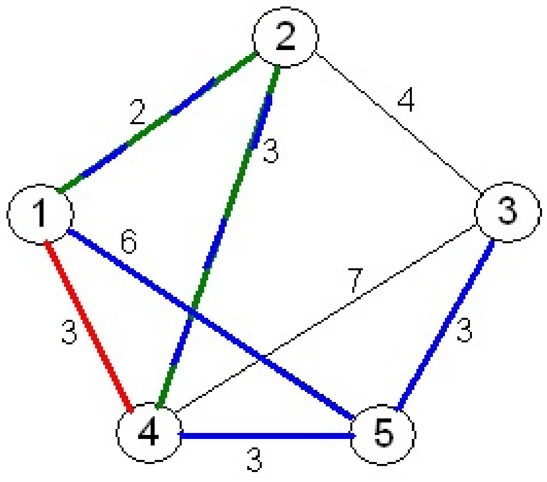

**Figure 1.1:** TSP tour is highlighted in blue.

Usually assume a **complete** graph (all possible $\binom{n}{2}$ edges exist). We can add missing edges with *high weight* to convert non-complete to complete.
Applications of TSP:

- School bus routes

- Delivery
- Tool path in manufacturing

To show the *decision version* of TSP (exist a tour of total weight $\leq k$) is NP-complete:

1. Show it is in NP (i.e. provide evidence (the tour itself) that there exists a TSP tour and show weights add up to $\leq k$)
2. Show a known NP-complete problem reduces in polynomial time ($\leq_p$) to TSP (the Hamiltonian cycle problem can be reduced to TSP)

## 1.3 Approach to NP-complete problems

For NP-complete problems we want to:

- Find exact solutions
- Find fast algorithms
- Solve hard problems

We can in effect only choose two: for hard problems we give up on either *fastness* (exponential time algorithms) or *exactness* (approximation algorithms).

## 1.4 Metric TSP

An appproximation exists for the **metric TSP** version, where:

- $w(u, v) = w(v, u)$
- $w(u, v) \leq w(u, x) + w(x, v) \quad \forall x$

An algorithm (1977) was proposed for metric TSP:

1. Find a minimum spanning tree (MST) of the graph
2. Find a tour by walking *around* the tree.

   Think of doubling edges of MST to get Eulerian graph (i.e. every vetex has even degree), which lets us find an Eulerian tour traversing every edge once.

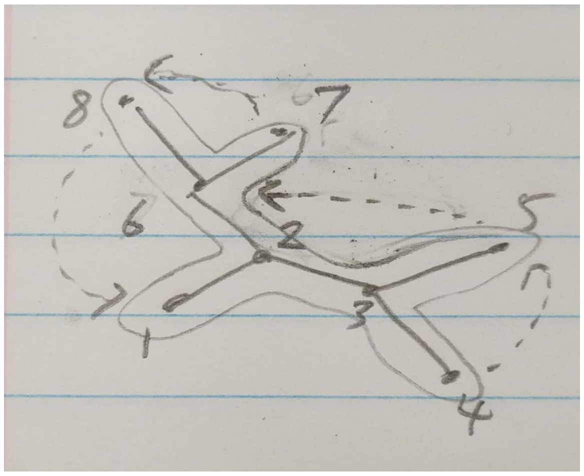

**Figure 1.2:** Eulerian tour is the solid line around the MST, the dotted lines show shortcuts taken, and the nodes are labelled in order.

3. Take shortcuts to avoid re-visiting vertices

   Instead of traversing a node twice (when walking around the MST), we take shortcuts and jump directly to the next unvisited node. By the triangle inequality our path should have a shorter path than if we actually traversed the MST edges twice, i.e.:

   $$l \le 2l_{MST}$$

   where $l$ is the length of our tour and $l_{MST}$ is the length of the MST (remember we doubled every edge).

   Note the total path length we get will differ depending on which node we start with: thus one must attempt all paths to find the best.

   **Lemma 1.1.** This algorithm is a **2-approximation**, i.e.:

   $$l \le 2l_{TSP}$$

   where $l_{TSP}$ is the minimum length of TSP.

   *Proof.* We need to show $l_{MST} \le l_{TSP}$.

   Take the minimum TSP tour. Throw out an edge. This is a spanning tree $T$. Since

   $$l_{MST} \le l_T \le l_{TSP}$$

   the result follows. □

   **Exercise 1.1.** Show factor 2 can happen.

Analyzing/implementing this algorithm (let $n$ # of vertices, $m$ # of edges):
Steps 2 and 3 take $O(n+m)$.
Step 1 is our bottleneck: we've seen *Kruskal's* (sorted edges with union find to detect cycles) and *Prim's* (add shortest edge to un-visited vertex) MST algorithms.
Prim's took $O(m \log n)$ using a heap. An improvement is using a *Fibonnaci heap* (1987) which improves runtime for MST to $O(m + n \log n)$. A further improvement uses a randomized linear time algorithm for finding the MST (1995).

**Theorem 1.1.** For general TSP (no triangle inequality) if there is a polynomial time algorithm $k$-approximation for any constant $k$, then $P = NP$.

*Proof.* Exercise (hint: start with $k = 2$ and the Hamiltonian cycle problem. Show the 2-approximation can be used to solve the HC problem). □

Can we improve factor of 2 for metric case? Yes (Christofides 1996):

1. Compute MST
2. Look at vertices of odd degree in MST (there will be an even number). Find a minimum weight *perfect matching* of these vertices.

   The MST and perfect matching is Eulerian: take an Eulerian tour and take shortcuts (as before).

Implementation: we need a matching algorithm - the best runtime (in this situation) is $O(n^{2.5}(\log n)^{1.5})$ (1991).

**Lemma 1.2.** We claim $l \leq 1.5 l_{TSP}$. Note that $l \leq l_{MST} + l_M$ (where $l_M$ is the total length of the minimum weight perfect matching). We must show that $l_{MST} \leq l_{TSP}$ and $l_M \leq \frac{1}{2} l_{TSP}$.
Sketch: to show $l_M \leq \frac{1}{2} l_{TSP}$, we show the smallest matching is $\leq \frac{1}{2} l_{TSP}$.

Open question: do better than 1.5 for metric TSP. We know the lower bound is 1.0045 (if we could get 1.0045-approximation then $P = NP$).
There is also the **Euclidean TSP** version where $w(e) =$ Euclidean length. We can get $\epsilon$-approximation $\forall \epsilon > 0$.