An Introduction to Complexity Theory and Approximation Algorithms

Grady Hollar

Fall 2025

1. An Introduction to Theoretical Computation

Introduction to computing. What is a Turing machine? What is an algorithm?

2. Decision Problems and Complexity Classes

What does it mean for problems to be in the classes P and NP? What are NP-complete problems?

3. Approximation Algorithms

Many problems of interest in computer science are those which ask us to produce optimal solutions. Some of these problems can be solved in polynomial time. For example:

Given an unweighted graph $G = (V, E)$ and two vertices $s, t \in V$,
what is the shortest path between s and t?

Readers who have taken a class on data structures and algorithms will know that a simple breadth-first-search on G will give the optimal solution in $O(|V| + |E|)$ time. Others, however, are not so easy:

Given a geographical map M, what is the minimum number of colors needed
to color M so that no two neighboring regions share the same color?

For problems like the one above, we currently have no hope in developing algorithms that would be able to produce optimal solutions efficiently. But what if we could get "close" to an optimal solution in polynomial time? This is the question that motivates the study of approximation algorithms.

Before defining such algorithms, we first need to formalize our notion of optimization problems.

Definition (Optimization Problem). An *optimization problem* Π is a pair (D_Π, f_Π), where D_Π is the set of *valid instances* for Π and f_Π is an *objective function*. Each instance $I \in D_\Pi$ has a nonempty set of *feasible solutions* $S_\Pi(I)$. The *size* of an instance, denoted by $|I|$, is the number of bits required to represent I in binary. Put

$$\mathcal{F}_\Pi = \{(I, s) : I \in D_\Pi, s \in S_\Pi(I)\}.$$

The objective function $f_\Pi : \mathcal{F}_\Pi \to \mathbb{Q}^+$ assigns a positive rational *cost* to each feasible solution of any given instance. Π is specified to either be a *minimization* or *maximization* problem. An *optimal solution* for an instance I of a minimization (maximization) problem is a feasible solution ω for I such that

$$f_\Pi(I, \omega) \leq f_\Pi(I, s) \qquad \Big(f_\Pi(I, \omega) \geq f_\Pi(I, s)\Big)$$

for all $s \in S_\Pi(I)$, and we denote the value of $f_\Pi(I, \omega)$ by $\mathrm{OPT}_\Pi(I)$.

Note/Aside: Implicitly, we will assume that all optimization problems we consider are **NP**-*optimization problems*. This notion is To ensure the optimization problems we consider are actually feasible to set up in the first place, we impose a few restrictions.

Definition (NP-optimization Problem). We say an optimization problem Π is an **NP**-*optimization problem* if each of the following hold:

(i) D_Π is recognizable in polynomial time.

(ii) For a fixed $I \in D_\Pi$, every $s \in S_\Pi(I)$ is of length polynomially bounded in $|I|$, and there exists a polynomial time algorithm that, given a pair (I, s), decides whether $s \in S_\Pi(I)$.

(iii) f_Π is computable in polynomial time.

Definition (Approximation Algorithm). Let Π be a minimization (maximization) problem, and let $\rho : \mathbb{N} \to \mathbb{Q}^+$ be a function with $\rho \geq 1$ ($\rho \leq 1$). An algorithm $\mathcal{A}$ is said to be a *ρ-factor approximation algorithm for* Π if, for every instance I, $\mathcal{A}$ produces a feasible solution s for I such that

$$f_\Pi(I, s) \leq \rho(|I|) \cdot \mathrm{OPT}_\Pi(I) \qquad \Big(f_\Pi(I, s) \geq \rho(|I|) \cdot \mathrm{OPT}_\Pi(I)\Big),$$

and $\mathcal{A}$'s running time is bounded by some polynomial in $|I|$.

3. The Traveling Salesperson Problem

Note. Throughout this section, we assume all graphs to be undirected and weighted.

Definition.

Optimization Problem (TSP). Given a complete, weighted, undirected graph $G = (V, E)$, what is the minimum cost Hamiltonian cycle in G?

Definition (Tree). A *tree* is a graph that contains no cycles.

Definition (Spanning Tree). Given a graph $G = (V, E)$, a *spanning tree* of G

Algorithm Approx-TSP(G, c)

Select an arbitrary $v \in G.V$ to be a "root"
Compute a minimum spanning tree T for G from root v using MST-Prim(G, c, v)
