Performance Ratios

Previously, we considered the issue associated with identifying a proving the complexity of hard computational problems. Much of that discussion focused on the class of NP-complete problems and emphasized their difficulty rather than their solution. Many real-world problems exists for which no known efficient solutions exist, because they are NP-complete or worse. Nevertheless, the need exists to solve this problems, at least to the point where they get reasonable solution.

When we consider "reasonable" solutions to hard problems, what we are really doing is looking for approximate solutions to these problems. Specifically, such approximate solutions are, in reality, "near-optimal" solutions to those problems. Generally speaking, the types of problems we will consider in this unit are optimization problems. Therefore, when we speak of finding approximate (or near-optimal) solutions to these problems, we will ultimately be concerned with how far from optimility the solutions are that are returned from the approximate algorithm. Often times, it is difficult to find an exact value; however, in many cases, it is possible to bound the approximation. One approach to doing so is through the derivation of a so-called ***approximation ratio***. Formally, an approximation algorithm is said to have approximation ratio $\rho(n)$ if for input size n, the cost c of the solution provided by the algorithm falls within a factor $\rho(n)$ of the optimal cost c^*. where

$$\max\left\{\frac{C}{C^*}, \frac{C^*}{C}\right\} \leq \rho(n).$$

As a matter of convention, an approximation algorithm that satisfies this condition is called a $\rho(n)$-approximate algorithm.

Recall that NP-complete problems have performance at best exponential in the input size (assuming $P \neq NP$). Thus, under the current constraints of computational complexity, any polynomial-time algorithm will necessarily yield a poorer solution. Even so, it is possible for many problems to associate solution quality to execution time. That is, for these problems, the longer they run, the better their solution. One class of algorithms with this behavior is the class of "anytime algorithms" where, after achieve an intiial level of performance, the algorithm can be halted at "any time" and return a solution. The longer the algorithm runs, the better the quality of the solution.

When considering the strategy for approximation, we define an ***approximation scheme*** to be an approximation algorithm that takes as input both the input size n and a target performance value $\epsilon > 0$. The associated algorithm would then would be a $(1 + \epsilon)$-approximate algorithm. As long as the associated algorithm runs in time that is polynomial in n, then the approximation scheme is said to be a ***polynomial-time approximation scheme***. If the algorithm is polynomial both in n and in $1/\epsilon$, then we also so it is a ***fully polynomial-time approximation scheme***.

In this unit, we will explore approximation methods by considering several NP-complete problems and their associated approximation algorithms. We will be apply approaches ranging from algorithms that are very problem-specific, to greedy algorithms, to linear programming-based methods. In each case, we will provide formal results on the performance of these algorithms.