Advanced Algorithms: Exercises

Panagiotis Karras
karras@comp.nus.edu.sg

1 Approximation Algorithms

We offer two sample exercises.

1.1 Sample Exercise 1

Suppose you are a consultant for the Port Authority of Singapore. As the busiest container port in the world, their revenue is constrained by the rate at which they can unload ships in the port.

A ship arrives with n containers of weight $w_1, w_2, \ldots w_n$. Standing at the dock is a set of trucks, each of which can load K units of weight. (You can assume that K and w_i are integers.) You can stuck multiple containers in each truck, subject to the weight restriction K; the goal is to minimize the number of trucks required to carry all containers. This problem is know to be NP-complete.

A greedy algorithm for this problem is the following. Start with an empty truck, and pile containers $1, 2, 3, \ldots$ into it until you get to a container that would overflow the weight limit. Now declare this track "loaded" and send it off; then continue the process with a fresh truck. This algorithm, by considering trucks one at a time, may not achieve the most efficient way to pack the full set of containers into an available collection of trucks.

1. Give an example of a set of weights, and a value of K, such that our algorithm does not use the minimum possible number of trucks.
2. Show that, however, the number of trucks used by our algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any value of K.

1.2 Sample Exercise 2

In the Load Balancing Problem, we are interested in placing jobs on machines so as to minimize the *makespan* - the maximum load on each machine. In some applications, it is natural to consider cases in which you have access to machines with different amounts of processing power, so that a given job may complete more quickly on one of your machines than on another. The question then become: How should you allocate jobs to machines in the more heterogeneous systems?

Suppose you have a system that consists of m *slow* and k *fast* machines. Fast machines can perform twice as much work per unit time as the slow machines. We are given a set of n jobs; job i takes time t_i to process on a slow machines and time $\frac{1}{2}t_i$ to process on a fast machine. We want to assign each job to a machine; as before, the goal is to minimize the makespan - i.e., the maximum, over all machines, of the total processing time of jobs assigned to that machine.

Give a polynomial-time algorithm that produces an assignment of jobs to machines with a makespan that is at most three times the optimum.

2 Randomized Algorithms

We offer two sample exercises.

2.1 Sample Exercise 1

Suppose we are given a graph $G = (V, E)$, and we want to color each node with one of three colors, even if we are not necessarily able to give different colors to every pair of adjacent nodes. Rather, we say that an edge (u, v) is *satisfied* if the colors assigned to u and v are different.

Consider a 3-coloring that maximizes the number of satisfied edges, and let c^* denote this number. Give a polynomial-time *randomized* algorithm that is expected to produce a 3-coloring that satisfies at least $\frac{2}{3}c^*$ edges.

2.2 Sample Exercise 2

Consider a very simple online auction system that works as follows. There are n *bidding agents*; agent i has a bid b_i, which is a positive natural number. We will assume that all bids b_i are distinct from one another. The bidding agents appear in an order chosen uniformly at random, each proposes its bid b_i in turn, and at all times the system maintains a variable b^* equal to the highest bid seen so far. (Initially b^* is 0.)

What is the expected number of times that b^* is updated when this process is executed, as a function of the parameters in the problem?

Example Suppose $b_1 = 20$, $b_2 = 25$, $b_3 = 10$, and the bidders arrive in the order $1, 3, 2$. Then b^* is updated for 1 and 2, but not for 3.