

15-750 — Graduate Algorithms — Spring 2006

Miller and Derryberry

Assignment 6 Due date: Friday, May 5, 2006.

Some Reminders:

- Read the Policies section on the course website before you start working on this assignment.
- You may work in **groups of size up to 3** for this problem set if you wish. However, **you should write up your solutions separately**. That is, collaboration should be limited to talking about the problems, so that your writeup is written entirely by you and not copied from your partner. In addition, state whether you worked alone and **list all collaborators**.
- Please refrain from consulting external materials when solving these problems. This does not apply to looking up standard inequalities, definitions, and such things (e.g., $(1 + 1/x)^x \leq e$ or Stirling's Formula).
- Please **submit both an electronic version, as well as a hard copy of your solutions** at the beginning of class on the due date.
- In all problems, it is implicit that you should show that your answer is correct, even when this is not explicitly stated.
- If you have questions, contact the course staff.

1 Maximum 2CNF

Let 2CNF denote the set of all boolean formulas in conjunctive normal form with at most two literals per clause. Let 2CNFSat be the subset of 2CNF formulas that are satisfiable. Define the set Max-2CNF as follows

$$\text{Max-2CNF} = \{(\phi, k) \mid \phi \in 2\text{CNF}, \text{ and } \phi \text{ has a truth assignment making at least } k \text{ clauses true}\}$$

- (a) (10 points) Show that the language 2CNFSat is polynomial time decidable.

SOLUTION: See Lecture 22 (“More on Reductions and NP-Completeness”) of Kozen.

- (b) (15 points) Show that deciding membership in Max-2CNF is NP-Complete.

SOLUTION: To show NP-completeness, we must show 2 things. First we must show that this problem is in NP, which is easy because we can prove that a particular instance (ϕ, k) is in Max-2CNF by giving a truth assignment that satisfies k clauses of ϕ .

Second we must show that Max-2CNF is NP-hard, which we can accomplish by giving a polynomial time reduction from vertex cover, $f : (G = (V, E), k) \rightarrow (\phi, k')$, as follows. For all $v \in V$, create the variable x_v and add the clause of the form \bar{x}_v to ϕ . For all $(u, v) \in E$, add n^3 copies of the clause $(x_u \vee x_v)$ to ϕ . Set $k' = |E|n^3 + n - k$ and return (ϕ, k') .

Clearly this reduction requires at most polynomial time. To see that it is valid, suppose G has a vertex cover of size k . Then by setting each x_v to be true if and only if it is in the vertex cover, we satisfy $|E|n^3$ “edge clauses” and $n - k$ “negated variable clauses.” Conversely, if we find a truth assignment satisfying k' clauses of ϕ , we choose exactly the vertices corresponding to the true variables to be in our vertex cover. This choice of vertices must be a vertex cover because all of the edge clauses must be satisfied, and it must contain at most k vertices because $n - k$ variables were set to false.

2 Randomized Move-To-Front

In class we considered an on-line algorithm problem called The List Update Problem.

In The List Update Problem we have a list containing a fixed set of n elements and a sequence of accesses to elements of the list. As in the notes, the cost of the operation $\text{Access}(x)$ is the index of x in the list. In addition, at any time, an algorithm is permitted to swap 2 adjacent members of the list at a cost of one per swap.

Consider the following on-line algorithm for The List Update Problem:

Random Move To Front (RMTF): After an access to an element x , flip a fair coin. If the outcome is heads, do a series of swaps to move x to the front of the list, otherwise do no swaps.

- (a) (15 points) Show that the expected competitive ratio for RMTF is 3 for the list-update problem.

Hint: Make sure you make use of random variable and linearity of expectation.

SOLUTION: Let A denote the list maintained by RMTF, and B be the list of an arbitrary off-line algorithm (also, for simplicity let B denote the algorithm as well where context suggests), where $A = B$ at the beginning of the sequence of requests.

During $\text{Access}(x)$, the amortized cost to RMTF is $\text{index}_A(x) + y \cdot \text{index}_A(x) + y\Delta\Phi$, where $\text{index}_A(x)$ is a random variable representing the index of x in list A , the cost to B is $\text{index}_B(x)$ (not a random variable), y is a $\{0, 1\}$ random variable indicating the result of the coin flip, and $\Delta\Phi$ is the change in our (as yet undefined) potential function if x is moved to the front of A .

To make our analysis easier, define S to be the set of elements in front of x in both A and B and T to be the set of elements in front of x in A but not in B (S and T are random variables). Thus, the amortized cost to RMTF is $1 + |S| + |T| + \Delta\Phi$ and the cost to B is at least $1 + |S|$.

Similarly to in class, let the potential Φ be 3 times the number of inversions between A and B . Thus, the amortized cost to RMTF is $1 + |S| + |T| + y(1 + |S| + |T|) + 3y(|S| - |T|)$.

Thus, the expected amortized cost of $\text{Access}(x)$ using RMTF is

$$\begin{aligned} E[A.C.] &= 1 + E[|S|] + E[|T|] + E[y(1 + |S| + |T|)] + E[3y(|S| - |T|)] \\ &= 1 + E[|S|] + E[|T|] + 0.5 + 0.5E[|S|] + 0.5E[|T|] + 1.5E[|S|] - 1.5E[|T|] \\ &= 1.5 + 3E[|S|] \\ &\leq 3 \text{ times the cost to } B, \end{aligned}$$

where in getting from line 1 to line 2 we used the independence of y from S and T . Also, note that each paid swap B does costs B 1 and costs RMTF at most an increase in potential of 3. Because the initial potential is 0 and remains non-negative, this proves that RMTF is 3-competitive in expectation to the optimal off-line algorithm since B was arbitrary.

3 Parallel Machine Scheduling

In the **parallel-machine-scheduling problem** we are given n jobs J_1, \dots, J_n with nonnegative processing times p_1, \dots, p_n for jobs, each to be scheduled on one of m identical machines. Once a job is started it must run until completion and no other job may be run on that machine until the first one is completed. Thus, a **schedule** is a map from jobs to pairs consisting of a start time and a machine such that no 2 jobs will overlap on the same machine. Let C_k be the completion time for the k th job. We define $C_{max} = \max_{1 \leq j \leq n} C_j$ to be the **makespan** of the schedule. The goal of the parallel-machine-scheduling problem is to schedule the jobs so as to minimize the makespan.

Consider three scheduling algorithms:

Greedy: Schedule the jobs in the order they are given and on the first available machine.

Shortest-First: Sort the jobs in non-decreasing order by their processing times. Run Greedy on this newly sorted list.

Longest-First: Sort the jobs in non-increasing order by their processing times. Run Greedy on this newly sorted list.

The goal of this problem is to determine how close these algorithms' makespans are to the optimal makespan. Let C_{opt} be the optimal makespan for an instance of the parallel-machine-scheduling problem. In order to determine the approximation quality of the schedules we need to find lower bounds on C_{opt} .

- (a) (2 points) Briefly show that $p_{max} = \max_{1 \leq k \leq n} p_k$, the maximum-sized job, is a lower bound on C_{max} .

SOLUTION: The optimal schedule must schedule J_{max} on some machine and that machine cannot finish before time p_{max} .

- (b) (2 points) Briefly show that $avg_m = \frac{1}{m} \sum_{1 \leq k \leq n} p_k$, the average machine load, is a lower bound on C_{max} .

SOLUTION: The heaviest-loaded machine in the optimal schedule must have a load that is at least equal to the average load.

- (c) (6 points) Show that Greedy and Shortest-First are 2-approximation algorithms for minimizing the makespan.

SOLUTION: It suffices to show that Greedy gives a 2-approximation because Shortest-First is a special case of Greedy. Suppose that the last job to complete, J_ℓ , starts being worked on at time t . Notice that $t \leq avg_m \leq C_{opt}$ and $p_\ell \leq p_{max} \leq C_{opt}$. Hence, the makespan, $t + p_\ell$ is at most $avg_m + p_{max} \leq 2C_{opt}$.

- (d) (10 points) Show that for any $\epsilon > 0$, Shortest-First is at best a $2 - \epsilon$ -approximation algorithm.

SOLUTION: Given ϵ , let $m = \lceil 1/\epsilon \rceil$ be the number of machines. Now, schedule $m(m-1)$ jobs of size 1 and one of size m . The optimal schedule load-balances perfectly and has makespan m while Shortest-First has a makespan of $2m-1$, which is $2 - 1/m \geq 2 - \epsilon$ times larger than C_{opt} .

- (e) (10 points) Show that Longest-First is a $3/2$ -approximation algorithm.

Hints: 1) Let J_t be a job with largest completion time in the Longest-First schedule. Show that if $2p_t \leq p_1$ the $C_{max} \leq 3/2 C_{opt}$.

2) Consider those problems where job J_t is scheduled by Longest-First at time zero.

SOLUTION: Let $J = \{J_1, \dots, J_n\}$ be the set of jobs, labeled in non-increasing order of size so that J_k is scheduled last. Suppose for the sake of contradiction that the resulting schedule has a makespan greater than 1.5 times the optimal makespan. Let J_ℓ be the last job to complete and suppose it starts at time t . Notice that $t \leq avg_m \leq C_{opt}$. Moreover, $p_\ell > \frac{t}{2}$, else $t + p_\ell \leq t + \frac{t}{2} \leq \frac{3avg_m}{2}$. Note that some machine must have at least 2 jobs on it from the set $\{J_1, \dots, J_\ell\}$ by the pigeonhole principle because $t > 0$ (if $t = 0$ then the schedule is optimal), so a lower bound on the makespan is $2p_\ell$ since p_ℓ is no larger than the m largest jobs. It follows that $t + p_\ell < 2p_\ell + p_\ell = 3p_\ell \leq 3/2 \cdot C_{opt}$. Contradiction.