Scheduling Theory **CO 454, Spring 2009**

Homework 2

Due: June 4th, 2009

1. **(3+3)**
   Suppose $f : \mathbb{R}_+ \to \mathbb{R}_+$ is a strictly increasing, positive function. That is, $f(x) > 0$ for all $x > 0$ and $f(x) > f(y)$ whenever $x > y$. Show by an interchange argument or otherwise that SPT gives an optimal schedule for the problem $(1||\sum_j f(C_j))$, that is, minimizing the sum of the function values of the completion times. Run SPT on the following data to output the schedule and the optimum $\sum_j f(C_j)$ for $f(x) = x^2$.

Jobs	1	2	3	4	5	6	7	8
p_j	3	4	1	5	2	6	2	3

2. **(6)**
   Consider the following data for minimizing average completion time with release dates.

Jobs	1	2	3	4	5	6
p_j	2	4	6	8	10	12
r_j	20	18	15	11	6	0

   Run the SRPT algorithm to get the optimal schedule for $(1|r_j, pmtn|\sum C_j)$. Use this to run the CFP algorithm done in class to get a schedule for $(1|r_j|\sum C_j)$. What is the ratio of the values of the two schedules? Find the optimal schedule for $(1|r_j|\sum C_j)$ in this case.

3. **BONUS (6)** Recall the algorithm CFP for $(1|r_j|\sum C_j)$. Let the schedule returned by the CFP algorithm have sum of completion times equal to CFP. Let the optimal schedule have sum of completion time OPT. We proved in class CFP was a factor-2 approximation algorithm, that is, $CFP \leq 2 \cdot OPT$. We say that the factor 2 is tight if for every constant $\delta > 0$, there is an instance of a scheduling problem such that $CFP \geq (2-\delta) \cdot OPT$. I asked in class if one could prove the factor 2 was tight for CFP.

   - Prove that if there are only 2 jobs, then $CFP \leq 1.5 \cdot OPT$.
   - For any $\delta > 0$, come with an example such that for that example $CFP \geq (1.5-\delta) \cdot OPT$.
   - Try to extend the example to many jobs to show for any $\delta > 0$, an example with $CFP \geq (2-\delta) \cdot OPT$, thus answering the question I asked in class.

4. **(6)** Run the LCL algorithm done in class for the problem $(1||f_{max})$ to find the optimal schedule for the following data

Jobs	1	2	3	4	5	6	7
p_j	4	8	12	7	6	9	9
$f_j(t)$	$3t$	77	t^2	$1.5t$	$70 + \sqrt{t}$	$1.6t$	$1.4t$

5. **(6)** Find a polynomial time algorithm to solve the problem $(1|prec|f_{max})$. Assume you are given a directed acyclic graph D representing the precedence constraints. What is the running time of your algorithm in terms of the number of jobs and number of arcs in D? Prove the correctness of your algorithm.
   (Hint: Think of the idea behind LCL of figuring out which job must be processed last. Try to use the same idea with precedence constraints. Take care that unlike for $(1||f_{max})$ *where any job could be processed last, in this case some jobs* cannot *be processed last. Which jobs are these? How will you modify the algorithm in presence of these jobs?)*

6. **(6)** In the knapsack problem we have n items, item j has profit p_j and weight w_j, and a knapsack of capacity B. The goal is to find the maximum profit subset of items whose total weight is at most B. In class we saw a dynamic program to solve the knapsack problem in time $O(nB)$. However, this algorithm was not a polynomial time algorithm if B was not polynomial in n. In this question we develop another dynamic program which runs in polynomial time if the maximum profit of an item, call it P_{max}, is polynomial in n (irrespective of the size of B).

   Construct the following table $T[i, p]$ which is supposed to contain the minimum weight subset S of items $\{1, 2, \ldots, i\}$ such that the profit of the items in S is *at least* p. If no subset of $\{1, 2, \ldots, i\}$ gives profit p, let $T[i, p]$ contain *null*. Maintain another table $t[i, p]$ which is supposed to contain the weight of items in $T[i, p]$. Let $t[i, p]$ be ∞ if $T[i, p]$ is *null*.

   (a) What is an upper bound on the maximum profit obtainable by any solution to the knapsack problem? (Answer can be in terms of P_{max}).

   (b) What is the size of the table you need to solve the problem? That is what should i range from? What should p range from? *(Hint: Use Q(a)).*

   (c) Suppose the table is constructed. How will you use the table to find out what is the optimum solution to the knapsack problem?

   (d) What are the smallest subproblems you can solve? What is $T[0, p]$ and $t[0, p]$, for any p? What is $T[i, 0]$ and $t[i, 0]$ for any i?

   (e) Suppose the table is computed so that $T[i, p]$ is known for all p in the range of p. How will you compute $T[i + 1, p]$? *(Hint: Consider the $(i + 1)$th item. What is $T[i + 1, p]$ if the $(i + 1)$th item is not in $T[i + 1, p]$? What is $T[i + 1, p]$ if the $(i + 1)$th item is there in $T[i + 1, p]$? How would you decide which is the case by using the $t[i, p]$ values?)*