Scheduling Theory **CO 454, Spring 2009**

Homework 3

Due: June 18th, 2009

Solutions

1. **(6)**
Run the dynamic programming algorithm done in class for $(1||\sum w_j U_j)$ for the following data.

Jobs	1	2	3	4	5
p_j	2	3	1	2	2
d_j	2	4	3	6	5
w_j	3	4.5	1	2	3

Solution:
Ordering the jobs accordingly to the due date we get $\{1, 3, 2, 5, 4\}$. The entries of the following table are the sets $T[i, q], t[i, q]$.

i / q	0	1	2	3	4	5	6
0	$\emptyset$, 0	$null, -\infty$	$null, -\infty,$	$null, -\infty$	$null, -\infty$	$null, -\infty$	$null, -\infty$
1	$\emptyset$, 0	$null, -\infty$	$\{1\}$, 3	$null, -\infty$	$null, -\infty$	$null, -\infty$	$null, -\infty$
2	$\emptyset$, 0	$\{3\}$, 1	$\{1\}$, 3	$\{1, 3\}$, 4	$null, -\infty$	$null, -\infty$	$null, -\infty$
3	$\emptyset$, 0	$\{3\}$, 1	$\{1\}$, 3	$\{2\}$, 4.5	$\{3, 2\}$, 5.5	$null, -\infty$	$null, -\infty$
4	$\emptyset$, 0	$\{3\}$, 1	$\{5\}$, 3	$\{2\}$, 4.5	$\{1, 5\}$, 6	$\{2, 5\}$, 7.5	$null, -\infty$
5	$\emptyset$, 0	$\{3\}$, 1	$\{5\}$, 3	$\{2\}$, 4.5	$\{1, 5\}$, 6	$\{2, 5\}$, 7.5	$\{1, 5, 4\}$, 8

2. **(6)**
In class, we saw a dynamic program to solve $(1||\sum w_j U_j)$ problem in time $O(n \sum_j p_j)$. Give a dynamic programming algorithm to solve the problem in time $O(n \log n + n \sum_j w_j)$. *(Hint: Construct a table with entries indexed by items and weight with $T[i, W]$ indicating a feasible subset of weight exactly W having minimum total processing time.)*

Solution: Maintain a table $T[i, W]$ which returns a feasible subset of jobs from $\{1, \dots, i\}$ with the minimum sum of processing times such that the total weight of the jobs is exactly W. If no such feasible set exists, we write $T[i, W]$ is *null*. We let $t[i, W]$ denote the sum of processing times of $T[i, W]$ with ∞ when the latter is *null*. Note that we are interested in finding the largest W for which $T[n, W]$ is not null. Once the whole table is constructed this can be done by going over the last row of the table.

Now we claim that $T[i + 1, W]$ can be computed from the smaller entries of the tables, that is, the problem exhibits an optimal substructure. To see this note that if the $(i + 1)$th job is indeed in $T[i+1, W]$, then since its due date is larger than all the other due dates in $\{1, \dots, i\}$, it will be processed last by the optimal schedule. Thus, the $(i + 1)$th job will not be late if and only if the sum of processing times of the remaining jobs of $T[i + 1, W]$ plus p_{i+1} is less than d_{i+1}. This is because we are storing the feasible subset which takes the minimum total processing time. Furthermore, $t[i, W - w_{i+1}] + p_{i+1}$ must be at most $t[i, W]$, for otherwise, $T[i, W]$ has smaller processing time. Therefore, the $(i + 1)$th job is in $T[i + 1, W]$ if and only if $t[i, W - w_{i+1}] + p_{i+1} \le d_{i+1}$ and $t[i, W - w_{i+1}] + p_{i+1} \le t[i, W]$. If the $(i + 1)$th job is not in $T[i + 1, W]$, then $T[i + 1, W] = T[i, W]$.

$$T[i+1, W] = \begin{cases} T[i, W - w_{i+1}] \cup (i + 1) & \text{if } t[i, W - w_{i+1}] + p_{i+1} \le d_{i+1} \text{ and } t[i, W - w_{i+1}] + p_{i+1} \le t[i, W] \\ T[i, W] & \text{otherwise} \end{cases} \tag{1}$$

Theorem 0.1. *The above dynamic program gives the optimal schedule for* $(1||\sum w_jU_j)$ *in time* $O(n\log n + n\sum_j w_j)$.

3. **(2+2+2)** For each of the statements, write true or false giving reasons.

   - If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.
   - If $X \leq_P Y$ and Y is NP-hard then X is NP-hard.
   - Let X be a problem in the class NP. If $P \neq NP$, then X cannot be solved in polynomial time.

   Solution:

   - True. Suppose Z can be solved in time $T(|Z|)$. As $Y \leq_P Z$, there exist two polynomials r, s such that Y can be solved in time $r(|Y|) + s(|Y|)T(|Y|)$. As $X \leq_P Y$, there exist two polynomials p, q such that X can be solved in time $p(|X|) + q(|X|)(r(|X|) + s(|X|)T(|X|)) = p'(|X|) + q'(|X|)T(|X|)$, where $p'() = p() + q()r()$ and $q'() = q()s()$ are polynomials. Hence $X \leq_P Z$.
   - False. The assertion shows that Y is harder than X, and not vice-versa, which is what we need. This is trivially reducible to any problem Y, but it is not NP-hard.
   - False, as $\emptyset \neq P \subseteq NP$. For example, consider an instance of $(1||\sum C_j)$ and the problem of determining if there exists a schedule of value at most B. This problem is in NP and can be solved in polynomial time, independently of whether P=NP or not.

4. **(3+3)**

   (a) The HPP (Hamiltonian path problem) is the following: given a graph G is there a simple path which contains every vertex of G. Recall that HCP (Hamiltonian cycle problem) was given a graph G, if there is a cycle containing each vertex of G. Show that $HCP \leq_P HPP$.

   Solution: We can assume that the number of vertices is greater than 4 for otherwise one can solve HCP in constant time. Suppose we have a polynomial time algorithm A for HPP. Given a graph G, we now show how to use A to get a polynomial time algorithm for HCP. If there is a hamiltonian cycle in G, then since there are at least 4 vertices for every vertex u there must be *distinct* vertices w, v, x such that $(w, u), (u, v), (v, x)$ are all edges of G and are in the hamiltonian cycle. Now look at the graph H obtained from G by deleting all edges incident to u and v except the edge (w, u) and (v, x). Thus, in the new graph H, the vertices u and v have degree 1. Now run the algorithm A on H. If there exists an hamiltonian path P, then since u and v are vertices of degree 1, they must be the end points. Thus the cycle $P \cup (u, v)$ is a hamiltonian cycle in G. Furthermore, if G has a hamiltonian cycle, then for at least one choice of w, v, x, the graph H would have a hamiltonian path.
   Algorithm:

   i. Pick any vertex u of G.
   ii. For all neighbor v of u, for all neighbors x of v $x \neq u$, and for all neighbors w of u such that $w \neq v, x$ (Note there are at most $O(n^3)$ (actually much better) iterations and thus this algorithm is polynomial time)
       - Construct graph H by deleting all edges in G incident to u and v except the edges (w, u) and (x, v).
       - Run algorithm A on H. If A returns yes, return yes.
   iii. If A returns no on all iterations above, return no.

   (b) In class we saw that $HCP \leq_P TSP$ which showed that TSP was NP-hard. Show that it is NP-hard to obtain a tour of total length at most βC^* for any $\beta > 0$, where C^* is the length of the optimal tour.

Solution: Given an instance of the HCP problem, construct an instance of TSP by setting distances $d(i,j) = 0$ if (i,j) is an edge, and $d(i,j) = 1$ if (i,j) is not an edge. In the YES instance of HCP, the optimum TSP solution is 0. Furthermore, any TSP solution of length 0 corresponds to a hamiltonian cycle. If there were a polynomial time algorithm which returned a tour of length at most βC^*, then in the YES case it would have still returned a solution of length 0. Therefore, the hamiltonian cycle instance is a YES instance if and only if the algorithm returned a tour of length 0. Thus $HCP \leq_P$ "β"-TSP.

5. (a) **(3)**
Show that the problem $(1|r_j|L_{max})$ is NP-hard by reducing it to the partition problem done in class. *(Hint: Given an instance of the partition problem, construct an instance of jobs with release dates such that if there is a partition no job is late, if there is no partition, at least one job is late)*

Solution Given an instance of partition: $\{a_1, \ldots, a_n\}$ such that $\sum_i a_i = 2B$, construct an instance of $(1|r_j|L_{max})$ with $n+1$ jobs. Jobs 1 to n have processing times $p_j = a_j$, release date $r_j = 0$ and due date $d_j = 2B+1$. Job $(n+1)$ has processing time $p_{n+1} = 1$, release date $r_{n+1} = B$ and due date $d_{n+1} = B+1$. We claim that there is a partition of the numbers $\{a_1, \ldots, a_n\}$ if and only if there is a schedule which finishes all jobs on time.

If there is a partition S of the numbers which add up to exactly B, then the schedule which processes jobs corresponding to S, then job $(n+1)$ and then jobs in $\{1, \ldots, n\} \setminus S$ will finish every job in time and start job $(n+1)$ at time B. In the other direction, if there is a schedule which finishes all jobs on time, then it must start job $(n+1)$ at time B and end it at time $(B+1)$. Furthermore, there cannot be any idle time as the due-dates of all other jobs is $2B+1$ which equals the sum of processing times. Thus, the jobs scheduled before job $(n+1)$ must have processing times exactly adding up to B. This will imply the partition.

(b) **(3)**
The above only shows that $(1|r_j|L_{max})$ is weakly NP-hard since partition is only a weakly NP-hard problem. Show that $(1|r_j|L_{max})$ is strongly NP-hard by reducing it to Bin Packing which is a strongly NP-hard problem.

Bin Packing: Given k items with sizes $(a_1, \ldots, a_k)$ and t bins each of capacity B, can one partition the items into the t bins such that the total size of the items in any bin is at most B.

(Hint: Given an instance of Bin Packing *construct an instance with $k + (t-1)$ jobs, where the first k jobs correspond to the sizes and the last $(t-1)$ jobs "partition" the jobs into bins.)*

Solution: Given an instance of bin-packing construct an instance of the machine scheduling problems with $n = k + (t-1)$ jobs denoted as $\alpha_1, \ldots, \alpha_k$ and $\beta_1, \ldots, \beta_{t-1}$. The processing time of job α_j is a_j and the processing time of job β_i is 1. The release dates of all α_i jobs is 0 and release dates of job β_i is $iB + (i-1)$. The due date of all α_i jobs is $tB + (t-1)$ and the due date of β_i is $iB + i$. We claim that there is a schedule which finishes all jobs in time if and only if the bin-packing instance has a feasible solution.

Suppose the bin-packing instance has a feasible solution, that is there exists a partition $J_1, \ldots, J_t$ of $\{1, \ldots, k\}$ such that $\sum_{j \in J_i} a_j \leq B$ for all $i \in \{1, \ldots, t\}$. Then we can schedule first the jobs α_j, for $j \in J_1$, then job β_1, then jobs α_j for $j \in J_2$, then job β_2 and so on. As $\sum_{j \in J_i} p_j \leq B$ for all $i \in \{1, \ldots, t\}$, each job β_i can start at its release time and end by the due date. Moreover, the last job α_j to complete will complete by time $(t-1)B + t - 1 + B = tB + (t-1)$.

Now suppose there is a schedule which finishes all the jobs in time. This implies that every job β_i is processed between time $iB + (i-1)$ and time $iB + i$. For every $i \in \{1, \ldots, t\}$ let J_i define the set of indeces of jobs α_j processed after β_{i-1} and before β_i. As there is no overlap between jobs, for every i we have $\sum_{j \in J_i} p_j \leq iB + (i-1) - [(i-1)B + (i-2) + 1] = B$, so $J_1, \ldots, J_t$ is a feasible solution to the bin-packing problem.