CSC375S **Problem Set 1 Solutions** **Winter/Spring 2005**

1. Consider the problem of *interval colouring* where the goal is to minimize the number of colours used in a feasible colouring where a feasible colouring is one in which intervals receiving the same colour do not intersect.

   Consider the following incompletely specified algorithm for the interval colouring problem:

   Sort the intervals $I \in \mathcal{I}$ in some way
   and let $I_1, I_2 \ldots, I_n$ be sorted sequence of input intervals.
   For $j = 1..n$
   Colour interval I_j with the lowest numbered colour c
   such that if I_k and I_j intersect for any $k < j$ then I_k did not receive colour c. EndFor

   (a) Say how to sort the intervals so that the above algorithm will be optimal; that is, will use the fewest possible colours.

   The intervals can be sorted so that $s_1 \leq s_2 \leq \ldots \leq s_n$ (or "equivalently" so that $f_1 \geq f_2 \geq \ldots f_n$.

   (b) Prove that your algorithm is optimal. Hint: what can be said at the time that the algorithm first uses it largest colour?

   Lets suppose that the algorithm uses c different colours and consider the interval (say I_j) that is the first interval (in the ordering above) to receive colour c. Then there is a set $\mathcal{I}'$ of $c-1$ intervals I_k with $k < j$ which have been assigned colours in $\{1, \ldots c-1\}$ which intersect with I_j. Since the intervals are sorted by non decreasing start times, we have $s_k \leq s_j < f_k$ for each $I_k \in \mathcal{I}'$ and hence these these $c-1$ intervals and I_j are all mutually intersecting.

   (c) Would the algorithm be optimal if the intervals were not first sorted? If yes, prove this fact; else give a counter example.

   No, the algorithm would not be optimal. In fact, no "online algorithm" can be optimal for interval colouring. Consider an input set whose first two intervals are $[1,3)$ and $[6,8)$. If the algorithm assigns the same colour $= 1$ to both intervals (as in the algorithm above) then let the next two intervals be $[4,7)$ and $[1,5)$. Since $[4,7)$ intersects $[6,8)$, it must be given a new colour $= 2$. And then since $[1,5)$ intersects both $[1,3)$ and $[4,7)$ it must be given a third colour. The optimal colouring uses 2 colours; namely, $[1,3)$ and $[4,7)$ get colour 1. If the algorithm gives the second interval $[6,8)$ a new colour $= 2$, then there is just one more interval $[2,7)$ which intersects both $[1,3)$ and $[6,8)$ and hence must get a third colour whereas the optimal clearly uses only 2 colours.

2. Consider the following "best fit" greedy algorithm for the unweighted interval scheduling problem on m identical machines.

   Sort $\mathcal{I} = \{I_1, \ldots, I_n\}$ so that $f_1 \leq f_2 \leq \ldots f_n$
   For $j = 1..m$
   $\quad t_j := 0 \quad$ % t_j will be the (current) latest finish time on machine j
   EndFor
   For $i = 1..n$
   $\quad$ If there exists j such that $t_j \leq s_i$ then
   $\quad\quad$ schedule I_i on that machine j which minimizes $s_i - t_j \geq 0$;
   $\quad\quad t_j := f_i$
   $\quad$ EndIf
   Endfor

   (a) Prove that "best fit" always produces an optimal schedule.

   The proof will use a "generic greedy optimality proof". Namely, we will show that the schedule σ_i after the i^{th} iteration is /it promising in the sense that it can be extended to an optimal solution. The proof is exactly like the proof for the one machine case except now (in addition to the other cases) we must say what we do (in the induction step) if best fit schedules the i^{th} interval on some machine j while OPT_{i-1} schedules this interval on machine $k \neq j$. We need an exchange argument to show that there is still an optimal solution OPT_i that extends σ_i. Let R_i^j (resp. R_i^k) denote the intervals in OPT_{i-1} scheduled on machines j and k with starting times $t \geq s_i$. We then simply convert OPT_{i-1} to OPT_i by moving R_i^j to machine k and moving R_i^k to machine j.

   (b) Consider the "first fit" greedy algorithm where we replace the statement "schedule I_i on that machine j which minimizes $s_i - t_j \geq 0$" by "schedule I_i on the machine of smallest index j such that $s_i - t_j \geq 0$".
   Show that "first fit" is not an optimal algorithm.

   Consider the 2 machine problem ISP_2. It is easy to come up with an example consisting of 4 intervals where the first two $[0, 2)$ and $[0, 4)$ overlap and get placed on machines 1 and 2. Then the third interval $[5, 6)$ gets placed on machine 1 which makes it impossible for "first fit" to schedule the fourth interval $[3, 6)$. In contrast "best fit" will schedule all 4 intervals.

   (c) What is the best approximation ratio you can prove for "first fit"?

3. Consider the following scheduling problem. We are given n jobs $J_1, \ldots, J_n$ where each job J_i is described by a processing time p_i and a value v_i. A schedule is simply an ordering π of the jobs so that $J_{\pi(1)}$ is scheduled first, $J_{\pi(2)}$ next, etc. The goal is to schedule all jobs sequentially (without overlap) so as to minimize the weighted completion time $\sum_{1 \leq i \leq n} v_{\pi(i)} * C_{\pi(i)}$ where $C_{\pi(1)} = p_{\pi(1)}$ and for $i \geq 2$, $C_{\pi(i)} = C_{\pi(i-1)} + p_{\pi(i)}$ is the completion time of the i^{th} job in the schedule. Give a (fixed order) greedy algorithm which always produces an optimal schedule. Prove that your algorithm is optimal.

Hint: Study the exchange argument used in section 4.2.

The greedy ordering is to sort the jobs so that $p_1/v_1 \leq p_2/v_2 \ldots \leq p_n/v_n$. We follow the exchange argument (as in section 4.2) to prove that that this is an optimal ordering. We consider an arbitrary optimal solution and show that any other ordering cannot be optimal. For suppose there is an optimal ordering which is not the greedy ordering. (For simplicity just assume that all the ratios p_i/v_i are distinct.) Then there is a pair of jobs which are scheduled so that J_i immediately follows J_k but $p_i/v_i < p_j/v_j$. Now exchange J_i and J_k and show that this reduces the objective function. Note that in computing the value of the objective function that only the values $v_i * C_i$ and $v_j * C_j$ change.

4. (a) Consider the unweighted interval scheduling problem (call it $MDEG$) and the third non-optimal greedy algorithm given in the text; namely in each iteration select an interval $I \in \mathcal{I}$ that intersects the fewest intervals in $\mathcal{I}$. Then remove that I and all the intervals it intersects. Give an argument showing that this algorithm is a 2-approximation algorithm.
   Hint: Want to show that when we choose an interval I, there can be at most 2 disjoint intervals intersecting I.

   As the hint suggests, we first show that when we chose an interval I, there can be at most 2 disjoint intervals intersecting I. For if not we would have to have some interval I' properly within I and not intersecting the other two disjoint intervals. Hence this interval I' would intersect fewer intervals than I. Now we can have a charging scheme whereby any interval in $OPT \cap MDEG$ is charged to $MDEg$ and any interval in $OPT - MDEG$ is charged to (say) the leftmost interval in $MDEG$ which it intersects which it intersects. By the hint, at most two intervals in $OPT - MDEG$ can intersect an interval in $MDEG$.

   (b) The text gives an example where this algorithm accepts three intervals whereas an optimal algorithm would accept 4 intervals showing that the approximation ratio is at best 4/3. Can you improve either the upper (2) or lower (4/3) bound for the approximation ratio of this algorithm?

   The optimal bound is 3/2. For the lower bound there is a sequence of examples $\mathcal{I}_m$ such that $\lim_{m\rightarrow\infty} OPT(\mathcal{I}_m)/MDEG(\mathcal{I}_m) = 3/2$. For the upper bound one can show a slightly stronger statement that
   $\forall \mathcal{I}\ 2 \cdot OPT(\mathcal{I}) \leq 3 \cdot MDEG(\mathcal{I}) - 1$.

5. Consider the following $\{0, 2, 3\}$ variant of the knapsack problem. Given n items $(p_1, v_1), \ldots, (p_n, v_n)$ and knapsack weight bound W, we define a feasible solution σ as a sequence $\sigma =< c_1, \ldots, c_n >$ in $\{0, 2, 3\}^n$ such that $\sum_i c_i * p_i \leq W$ and define its value $V(\sigma)$ as $\sum_i c_i * v_i$. That is, every item that occurs in the knapsack occurs 2 or 3 times.

   Describe a dynamic programming algorithm for this $\{0, 2, 3\}$ knapsack problem by defining an appropriate "semantic array" A , and a recurrence (including the base cases) for computing elements of A. Justify why the recurrence is correct.

Let $S(\sigma)$ denote the intervals such that $c_i > 0$. The array we want to compute is $OPT[i, t] =$ the maximum profit of a feasible solution σ with $S(\sigma) \subseteq \{1, \ldots, i\}$ and $\sum_{i \in S(\sigma)} c_i * p_i \leq t$.
The recurrence is :
$V[0, t] = 0 \ \forall t$
$V[i, t] = \max\{V[i-1, t], V[i-1, t-2p_i] + 2v_i$ (if $2p_i \leq t$), $V[i-1, t-3p_i] + 3v_i$ (if $3p_i \leq t$)$\}$