Homework 2

Michael Wurtz and Frederick Robinson

20 January 2010

1 Problem 4.4

We are given two ordered sets $S' = \{s'_1, \ldots, s'_m\}$ and $S = \{s_1, \ldots, s_n\}$. We want to test whether S' is a subsequence of S.

1.1 Algorithm

1. Set two counters $k = 1$ and $l = 1$.
2. While $k \leq m$ and $l \leq n$
   (a) If $s'_k = s_l$
      i. If $k = m$ then output "Subset" and end.
      ii. If $k \neq m$ then increment k
   (b) Increment l
3. Output "Not a Subset"

Intuitively, the algorithm just looks at the items of S in order until it matches the first item of S', then continues looking down the list of S until it matches the next value of S'. This is repeated until all values in S' are located, or it is determined that this will not happen.

1.2 Correctness

The algorithm always outputs an answer since, if it reaches the end it outputs "Not a Subset" and the only way for it to terminate without reaching the end is to output "Subset." Moreover, it must terminate, because in every run of the loop it increments either l or k, and it ends when one of these grows past a fixed value.

No False Negatives If S' is a subsequence of S then it will be identified as such. The only way that the algorithm returns "Not a Subset" is if it does not find the last element of the subset in the list.

Additionally the only way that this could happen with the last element actually in the list is if it started looking in S after it had already passed the

corresponding element. However, this would mean that the subset is misordered, which is a valid reason for saying that it is "Not a Subset"

No False Positives If the algorithm terminates, having found the last element of S' it must have found each previous element. That is, if the algorithm finds element s'_i, it must have found s'_{i-1} as that is the only way the k counter would have been incremented to i. By induction, if it terminates having found s'_n, it must have found each s_i with $i < n$. Also, as the l counter only increases, the elements of S' in S are ordered correctly.

1.3 Runtime

The l counter is incremented in each iteration of the while loop, so the loop runs at most n iterations. As one iteration of the loop is constant time, the algorithm is $O(n)$

2 Problem 4.7

Given $J_1, \ldots, J_n$ where $J_i = (p_i, f_i)$: Order the jobs by f_i, starting with largest f_i. This is an optimal scheduling of the jobs.

2.1 Runtime

Comparison sort of n elements is $O(nlog(n))$

2.2 Correctness

Sketch of Correctness Proof:

First we prove that in a job schedule, swapping two adjacent jobs cannot hurt the schedule's time if the first job's parallel time is not more than the second's. Then we show that our algorithm's schedule can be reached by these non-harmful swaps from any arbitrary schedule. So in particular our schedule can be reached from any other optimal schedule without increasing the schedule's time.

Correctness:

Lemma 2.1. *If $J_1, \ldots, J_n$ are sorted into a schedule, and there is a J_i with $f_i \leq f_{i+1}$, then swapping J_i with J_{i+1} will not decrease the total runtime of the jobs.*

Proof. In the first ordering, suppose J_i starts at time t. Then J_{i+1} starts at $t + p_i$, and J_{i+2} starts at $t + p_i + p_{i+1}$. This means J_i finishes at $t + p_i + f_i$, and J_{i+1} finishes at $t + p_i + p_{i+1} + f_{i+1}$. In the swapped ordering, J_{i+1} will start at t. Then J_i starts at $t + p_{i+1}$, and J_{i+2} starts at $t + p_{i+1} + p_i$. This means J_{i+1} finishes at $t + p_{i+1} + f_{i+1}$, and J_i finishes at $t + p_{i+1} + p_i + f_i$. All tasks before J_i will finish at the same time regardless of whether or not J_i and J_{i+1} are swapped. As J_{i+2} starts at the same time regardless of the

swap, it and all tasks after it are also unaffected by this swap. As $f_i \leq f_{i+1}$, $t + p_{i+1} + p_i + f_i \leq t + p_i + p_{i+1} + f_{i+1}$, and $t + p_{i+1} + f_{i+1} \leq t + p_i + p_{i+1} + f_{i+1}$. All jobs in this reordering finish at the same time or earlier than a job in the original ordering, so the new ordering cannot take more total time than the old ordering. □

Now, take any arbitrary scheduling. Perform a bubble sort[1] on the scheduling to move jobs with larger f_i earlier. As a bubble sort only swaps adjacent pairs, and only if it is smaller than the next job, none of the swaps a bubble sort does will cause the total time to become longer. Next, if there are terms with equal parallel times, they may also be rearranged in any order by the lemma without hurting the scheduling. Thus, the scheduling that was sorted by f_i is never worse than any arbitrary scheduling, and is optimal.

3 Problem 4.8

A connected graph G with unique edge costs must have a unique minimum spanning tree.

Proof. Suppose towards a contradiction that there is a graph with unique edge lengths which does not have a unique minimum spanning tree. Then there exist at least two spanning trees with the same (minimum) value. Call one of these trees T and the other T'. There must exist some edge e in T' which is less in value than the maximal edge of T and is not a part of the tree T.

If such an edge does not exist, then each edge in T' is less than or equal to each edge of T, but this is inconsistent with T and T' having the same length and edge length being unique.

Consider what happens when we add e to T. A cycle must form, as before we had a spanning tree. Call this cycle $\{e, n_1, \ldots, n_k\}$.

Case 1: The cycle has an edge which is greater in length than e. Remove this longer edge from the graph keeping the rest of T fixed, and we get a tree: any path that went through that edge before is now re-routed around the cycle, so all nodes are still connected. This tree was formed by changing an edge for a shorter edge, thus it has a smaller length and the previous tree was not a MST. Contradiction.

Case 2: The cycle does not have an edge which is greater in length than e. Then each $\{n_1, \ldots, n_k\}$ is less in value than e by uniqueness. If we remove e from T' we divide T' into two trees. We know that the nodes which are part of the cycle in T are divided into members of two separate subtrees in our cut T'.

If not, then the nodes that were connected by e were also connected by some other path, and T' was not a tree.

Because the members of the cycle in T are parts of both subtrees in T' one of $\{n_1, \ldots, n_k\}$ connects these two subtrees.

[1] this is not used in the algorithm due to slow runtime, but can still be used to illustrate correctness of any sort

Inserting this edge and removing e from T' will decrease the length of T, a contradiction of T' being a minimum spanning tree. □

4 Problem 1

4.1 Matroid

Interval scheduling is not a matroid because it fails to satisfy the augmentation property. There exist two feasible choices for a schedule, one schedule with more tasks than the other but where there is no task of the larger set which can be added to the smaller without creating an invalid schedule.

For example, one feasible schedule is to have one task from 2 to 5. Another feasible schedule is two tasks; one from 1 to 3, and the next from 4 to 6. The second set has more elements, but none of those elements may be added to the first set without creating a time conflict.

The interval scheduling problem is therefore not a matroid.

4.2 Greedy

The greedy by value algorithm is not optimal for interval scheduling with values. Consider this situation: The task with the highest value takes up all available time, and there are multiple less valuable tasks which, taken together, sum to a higher value than the one task. The greedy by value algorithm will pick the most valuable task, while it should select all the small tasks.

For a specific example, take the task collection made up of

1. time 0 to 5 with value 2
2. time 0 to 1 with value 1
3. time 2 to 3 with value 1
4. time 4 to 5 with value 1

Greedy by value will select the value 2 task first, locking out all others. However, the last 3 tasks can be taken together for a total value of 3, which is the optimal solution.

5 Problem 2

Suppose we have a non-matroid system. Then either the subset property or augmentation property is false. We will demonstrate that if the system lacks either of these properties then greedy by value does not work, thus showing that greedy by value works only if the system is a Matroid.

5.1 Assume the subset property is false.

There exist sets A and B so $A \subset B$ with B a feasible set and A not a feasible set.

For a counterexample, assign a value of 1 to all elements of A and 0 to all other elements of the ground set.

The greedy by value algorithm will order the elements by value; all elements of A will be before any other element. So the algorithm will proceed down the list, and after reaching the end of all elements in A, not all elements of A will be in the accumulation set. This is because the accumulation set is always feasible, but A is not feasible. The algorithm will then continue down the list, perhaps adding other elements with value 0, but it will never return and select elements of A it skipped.

The total value of the collection of elements made is $1 \cdot n = n$, where n is the number of elements of A chosen. However, the set B has total value $1 \cdot m = m$, where m is the number of elements of A ($m > n$). The collection given by the greedy algorithm is not optimal.

5.2 Assume augmentation is false

Even given our assumption the subset property may still be true. We'll assume that it is true for the purposes of this argument because, after all if it is not we're done already by the previous. Specifically, assume there exist feasible sets A and B with $|A| < |B|$, but there is no element of B which is not already an element of A but that can be added to A to give a feasible set.

Let $n = |A \setminus A \cap B|$, the number of elements unique to A. Also, let $m = |B \setminus A \cap B|$, the number of elements unique to B. We know that m must be nonzero since $|A| < |B|$.

For a counter example, assign values to the elements of the ground set as follows:

1. $\frac{m+n}{2n}$ to all elements in A. $m > n$ so this is greater than 1
2. 1 to all elements in $B \setminus A$
3. 0 to all elements of the ground set in neither A nor B

The greedy by value algorithm will first select all elements of A as these have the highest value. All of these elements will be successfully added as the subset property guarantees feasibility. The third highest values are the elements of B, but by the assumption, none of these can be added to our set while still being feasible. Next, other elements of the ground set will be checked for feasibility, but as they have values of 0, it does not matter whether any are added or not.

The total value of the set greedy makes is $\frac{m+n}{2n}|A \cap B| + \frac{m+n}{2n}n = \frac{m+n}{2n}|A \cap B| + \frac{m+n}{2}$

However, B is a feasible set with total value $\frac{m+n}{2n}|A \cap B| + m$, and as $m > n$, $m = \frac{m+m}{2} > \frac{m+n}{2}$, B has higher value than the set the greedy by value algorithm made, so we are done.