CSCC73

1 Greedy Algorithms

Definition – Optimization Problem

An optimization problem is one where we want to find the min or max of a given objective function subject to constraints. The min/max is called the **optimal solution**. Note that there often can be multiple.

Definition – Greedy Algorithm

A greedy algorithm is one that builds up the solution in stages, iteratively working on the previous stage's partial solution by judging what is going to likely lead to a more correct next solution.
A greedy algorithm is what we say is **greedy** and **irrevocable**. Greedy referring to how the above process is very myopic and doesn't look past the immediate next stage, and irrevocable means that we do not backtrace, what is done is done.

Greedy algorithms are commonly used to solve optimization problems. However, greedy is not always possible. A greedy algorithm is definietly not guaranteed to work, but when it does, it can be very fast because of the greedy and irrevocable properties of the algorithm.

1.1 Example Optimization Problem applying a Greedy Algorithm

Definition – Interval Scheduling

An Interval Scheduling problem is one where we have n jobs, each has a start time defined by $s(j_n)$ and an end time defined by $f(j_n)$ with $s(j_n) \leq f(j_n)$. The problem is a maximizing problem. What is the largest set of non-conflicting jobs can we have? A set that satisfies this condition is called a **feasible set**.

Definition – Job Time Interval

A job time interval for a job j_i is defined as $\Big[s(j_i), f(j)\Big)$

Definition – Conflicting Job

A conflicting job is one where the interval of one job overlaps with another.

An example of an interval scheduling issue is the lecture hall. We have n lectures that we want to schedule with varying durations, start times and end times. So to make the most of the lecture hall, how can we best schedule each lecture to maximize the number of lectures that happen on a given day?.

Max Cardinality

We are looking for the largest possible set of non-conflicting issues. This means that the set cannot be extended. However, being non-extendible does NOT mean that we have the largest set. Its only a one way implication. It is easy to imagine how.

Given our pseudocode for a scheduler, we will always get a set that is non-extendible. But how do we know

if it is the largest? Well the thing is that we do not always get the largest. So how can we modify the algorithm to ensure this? One trick that we can pull is sorting our data beforehand. But how should we do this?

1.1.1 Sort jobs by earliest start time

So what if we sort jobs, in ascending order, by their start time? This is easily disproved as a bad idea. Imagine we have one mega long job, and we have a bunch of mini jobs, where none start before the mega long job.

1.1.2 Sort jobs by shortest job interval

This is also easily disproved as a bad idea. To find a counter example, just take two long jobs that do not conflict, then put a shorter job that conflicts with both.

1.1.3 Sort jobs by smallest number of job interval conflicts

This one is also not optimal. It requires a slightly more complex counter example.
Note that this option is also dependent on the order we pick. The counter example chosen illustrates this.

1.1.4 Sort jobs by finish time

This option will be guaranteed to produce an optimal outcome.

1.2 Proving that sorting by finish time is optimal

There are two approaches to this proof. Either we can use the "promising set" method or the "greedy always ahead" method. Note that both of these proofs prove some kind of loop invariant. It is then up to us to show that the partial solution is indeed correct, and then show that our loop ends.

Theory – Promising Set Greedy Algorithm Proof

The promising set proof method for a greedy algorithm is one where you aim to show that the partial solution (and ultimately the solution generated by the last stage) are all subsets of an optimal solution. This should be proved via induction.

Theory – Greedy Always Ahead Algorithm Proof

The Greedy Always Ahead Algorithm proof involves trying to show that the algorithm is always "doing the right thing ". This should be proved via induction.

Information – Soft by Finish Time INTSCHED

Pseudocode:
Running Time:
If we pick a competent sorting algorithm, we can finish the sort in $\mathcal{O}(n \log n) + \mathcal{O}(n) = \mathcal{O}(n \log n)$

1.2.1 Promising Proof

Lemma 1 (bruh):

W

e say "promising", because here, we want to prove that every partial answer is a subset of an optimal answer. First we will need the promising set lemma, which goes as follows:

Lemma: Promising set $\forall$ iter $t\exists$ opti set $A_t^* \ni$: $A_T \subseteq A_t^*$
In regular English, this means for each iteration t, it's partial solution is a subset of an optimal solution.
We will prove this via induction.
Base case: Empty set. This is trivial, the empty set is a subset of all sets.
Induction set: Assume $A_t \subseteq A_t^*$, we will show that $A_{t+1} \subseteq A_{t+1}^*$

We will break $t+1$ into two cases. First, what if j_{t+1} is in conflict with another job? In which case we just say that $A_{t+1} = A_t$ and we are done.
Next we will consider if we have to include j_{t+1}. In this case we will break it down further into two sub cases. First, what if j_{t+1} IS part of A_t^*? Then in this case $A_t^* = A_t^*$.
If $j_{t+1} \notin A_t^*$ is a bit more complex. This means that j_{t+1} is in conflict with some job in A_t^*. If that is the case then we want to show that j_{t+1} is ONLY in conflict with one job in the A_t^*. If it is, then we can swap it out. Otherwise we are in some deep shit. So lets consider, what if there is more than one conflict? Well then that goes against our entire premise since we have the set of jobs sorted by earliest finish time. This would mean that the first job that j_{t+1} is in conflict with would actually have been considered BEFORE j_{t+1}, and we wouldn't have this situation to begin with. As such this case is invalid. So that means j_{t+1} is only in conflict with one job called j'. And as such we toss out j' and say: $A_{t+1}^* = \left(A_t^* - \{j'\}\right) \cup \{j_{t+1}\}$ and $A_t \cup \{j_{t+1}\} \subseteq A_{t+1}^* = A_{t+1}$. $\square$

1.2.2 Greedy-Always-Ahead Proof

For this method, we will change our invariant to the following:
Lemma: $\forall i \in [1, \ldots, k], f(j_i) \leq f(j_i^*)$
Proof:
Base case: Trivial.