Greedy Algorithms

Frank

September 18, 2021

1 Greedy Algorithms

1.1 Problems Applicable by Greedy Algorithm

- **Optimization Problems** : Given an input, find *solution* that
  - Satisfy constraints
  - Optimizes (max or min) objective function

  For example, given a graph, we can find the shortest path between two nodes. The constraint then is the verticies must form a path and we are optimizing for the length of the path.

1.2 Characteristics of a Greedy Algorithm

- Builds solution incrementally in ***stages***, typically in a *for loop*
- At each stage, ***extend*** current partial solution to a fuller solution greedily and irrevocably
  - ***Greedy*** means that the algorithm will take the best choice at every stage with no regard for the future. Looks one step at a time.
  - ***Irrevocable*** means that you cannot go back on your choices, once you've made the decision, you will continue from there. Cannot undo past choices.
- Very efficient when applicable to a problem
- The key part in the design of a greedy algorithm is deciding how to extend the partial solution into an optimal solution

2 Interval Scheduling (KT 4.1)

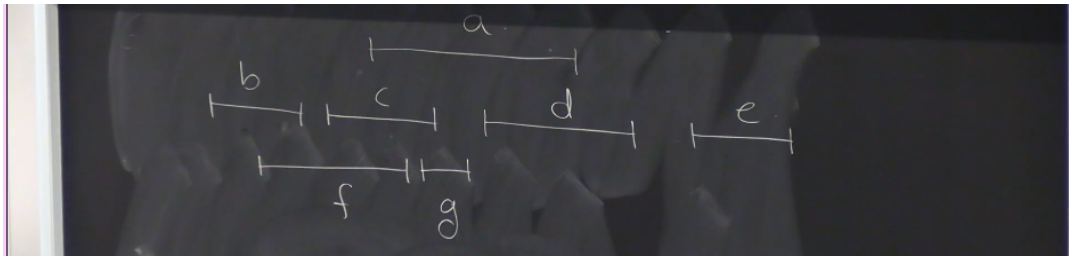
2.1 Problem

Input:

- Set of n intervals.
- Each interval i has start time $s(i)$ and finish time $f(i) > s(i)$. $\implies [s(i), f(i)]$

- $i\&j$ overlap if $[s(i), f(i)]\&[s(j), f(j)]$ have more than one point in common
- Set of intervals is feasible if no two intervals overlap

- Consider input: Let objective function be the number of intervals in the set. Want to maximize this objective function. With the constraint being that the set of intervals is feasible.



Output: A max cardinality feasible subset of input

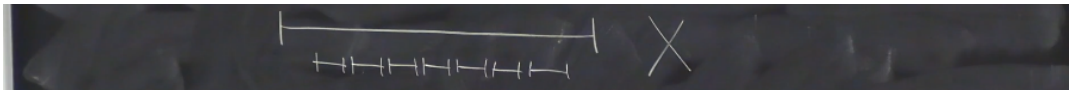
2.2 Potential Greedy Algorithms

```
1: procedure GREEDY
2:     sort intervals in some order
3:     A := ∅
4:     for (each interval i in sorted order) do
5:         if (i overlaps no interval in A) then
6:             A := A ∪ {i}
7:     return A
```

We mentioned in some order, but in what order?

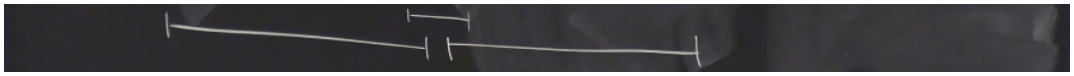
2.2.1 In what order?

1. Sort by start time (most recent)
    - This is not a good way, a counter example is



2. Sort by length (shortest)

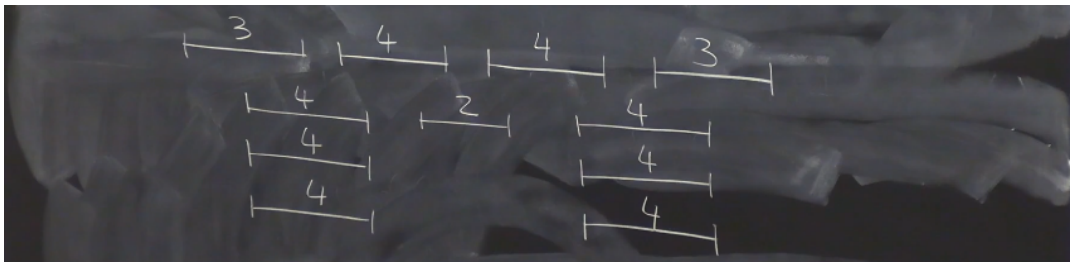
   - Better but not optimal, a counter example is



   - Note that 2 is a lot better than 1. Since 1 can be arbitrarily bad, meaning that we can keep adding smaller intervals inside of the bigger interval that begins first to make it increasingly worse.
   - However, the worst 2 can do is choosing one interval instead of two intervals.

3. Sort by number of overlaps (fewest)

   - Very complicated, yet not the optimal, a counter example is



   - Note the best we can do here will be 3 intervals instead of the optimal (4)

4. Sort by finish time (most recent)

   - Optimal solution.

     Intuition: Taking the most recently finished interval leaves the most room for other intervals to be included in A

First, consider the Running time of this algorithm

2.2.2 Running time

```
1: procedure GREEDY
2:     sort intervals in increasing finishing time        ▷ O(n log n)
3:     A := ∅
4:     for (each interval i in sorted order) do           ▷ O(n^2) or O(n)
5:         if (i overlaps no interval in A) then
6:             A := A ∪ {i}
7:     return A
```

- The above algorithm runs in $O(n^2)$
- Note that we can improve the $O(n^2)$ to $O(n)$ by keeping track of the finishing time of the previous interval in A then compare that to the start time of the upcoming chosen interval i

Which yields

```
1: procedure GREEDY
2:     sort intervals in increasing finishing time                ▷ O(n log n)
3:     A := ∅
4:     for (each interval i in sorted order) do                   ▷ O(n)
5:         if (s(i) ≥ Finish time of previous interval in A) then
6:             A := A ∪ {i}
7:     return A
```

- $O(n \log n)$ algorithm

2.2.3 Proof of optimality 1 (Greedy-stays-ahead)

Claim 1 : A is feasible (trivial)

Let $A = j_1, j_2, \ldots, j_k$ in L-R order, which is unambiguous due to A being feasible

Let $A^\star$ be any optimal solution, $A^\star = j_1^\star, j_2^\star, \ldots, j_m^\star$ in L-R order, where $k \leq m$ since by default, m should be the largest since it is optimal. We want to prove that $k = m$.

(*Note that $j_1, \ldots, j_k$ was added to A in L-R order, but we cannot claim anything about what order $j_1^\star, \ldots, j_m^\star$ was added into $A^\star$*)

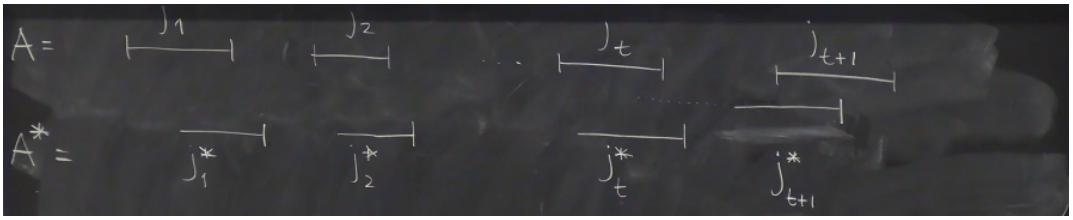
Claim 2 (greedy-stays-ahead): $f(j_t) \leq f(j_t^\star) \quad \forall t = 1, \ldots k$. This implies that for every step, the greedy algorithm chose the interval that finishes before the interval chosen by the optimal algorithm, thus providing more room for future intervals to be added into the set (Then if optimal is able to choose an interval, so will greedy). If such a claim is true, then we know that greedy is at least as good as optimal, since *both algorithms acts upon the same original set of intervals.*

Proof by induction on t

Basis: $t = 1$, $f(j_1) \leq f(j_1^\star)$ by step 2 of algorithm, which places the interval with the most recent finish time first to be added into A. HOLDS.

IS : Suppose true up to t. Prove also true for $t + 1$

Prove by contradiction



Assume $f(j_{t+1}) > f(j_{t+1}^\star)$ ... (add proof here), arriving at contradiction.

Now we use our claims to proof that A is indeed optimal

Thm : The set A returned by the greedy algorithm is optimal

Pf: Must prove that $k = m$ or $\|A\| = \|A^\star\|$

Suppose for contradiction, $k \neq m \implies k < m$

Then $j^\star_{k+1}$ exists

And by Claim 2, we have that $f(j_k) \leq f(j^\star_k)$

Now consider $f(j_k) \leq f(j^\star_k) \leq s(j^\star_{k+1})$ (feasible, since is optimal)

Finally $f(j_k) \leq f(j^\star_k) \leq s(j^\star_{k+1}) < f(j^\star_{k+1})$, implying that the greedy algorithm should have considered $j^\star_{k+1}$ since it finishes later than j_k and does not overlap, but does not (since it stops at j_k). This contradicts our assumption that A has k elements. This means that it must be true that $k \geq m$.

$\therefore A$ is optimal

2.2.4 Proof of optimality 2 (Promising Set)

A_t is the set contained in A at end of iteration t

Claim 3: $\forall$ iteration t $\exists$ optimal solution $A^\star_t \supseteq A_t$. Note that for every iteration, our optimal solution may change, since we are not claiming "$\exists$ optimal solution $\forall$ iteration $A^\star_t \supseteq A_t$", which is a much stronger statement.

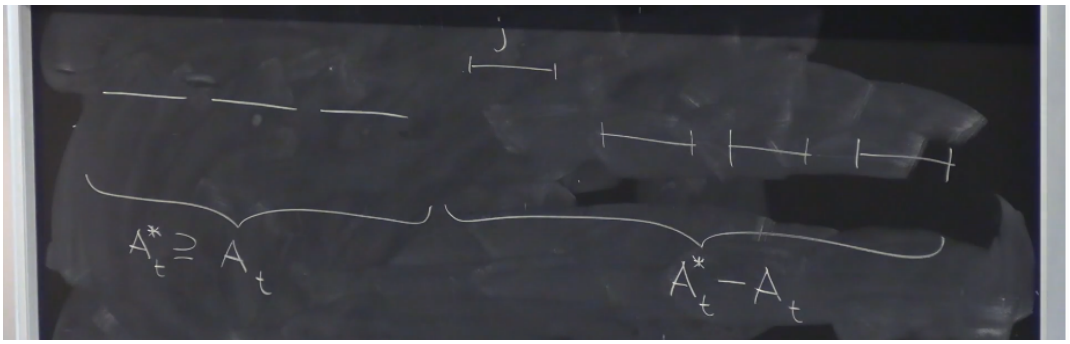
Proof by induction on t

Basis: $t = 0$ Holds, since A is empty and therefore is by default contained in some optimal set

IS: Assume true for t, $\exists$ optimal set $A^\star_t \supseteq A_t$ Prove for $t+1$: $\exists$ optimal set $A^\star_{t+1} \supseteq A_{t+1}$

Case 1: $A_{t+1} = A_t$, nothing was added to A. Since A has not changed over the iteration, then $A^\star$ has not changed. So take $A^\star_{t+1} = A^\star_t$, then $A^\star_{t+1} \supseteq A_{t+1}$ as wanted.

Case 2: $A_{t+1} = A_t \cup \{j\}$.
If $j \in A^\star_t$, then we have chosen an interval already in $A^\star_t$, then we do not need to change $A^\star_t$, so again $A^\star_{t+1} = A^\star_t$, then $A^\star_{t+1} \supseteq A_{t+1}$ as wanted.
If $j \notin A^\star_t$, consider the following picture:



We have that all unchosen intervals in $A_t^\star$ are in future iterations. We will add a new interval j into A_t. Note that j must overlap with another interval in $A_t^\star$ since if it does not overlap, we get a set with greater cardinality than the optimal set, which is not possible. So $\exists j^\star$ that is interchangeable with j. So take $A_{t+1}^\star = (A_t^\star - \{j^\star\}) \cup \{j\}$ which is an optimal set that contains j and thus contains A_{t+1} $\square$

Proof of Thm using Claim 3 (instead of Claim 2)

Claim 3 $\implies A_n \subseteq A_n^\star$

Will show that $A_n = A_n^\star$

Suppose on the contrary that $\exists j \in A_n^\star - A_n$, then j does not overlap any interval in $A_n^\star$ (other than itself), then it follows that j does not overlap any interval in A_n, which is impossible because then the algorithm would choose j. Thus we have a contradiction.

$\therefore A_n = A_n^\star$ and A is optimal $\square$

3 Additional Related Problems

- **Weighted Interval Scheduling**: Same problem except now the interval is weighted. We want a set where the sum of the weight of all the intervals in the set is the largest instead of the largest cardinality of the set.
- **Interval Partitioning**: Instead of a single resource, we have multiple resources which can contain the intervals. We want to minimize the number of resources used to contain all of the intervals in a feasible manner.