CS 341 Algorithms. Spring 2025. University of Waterloo

Lecture 8: Greedy algorithms for scheduling problems

We start our study of greedy algorithm using scheduling problems as examples.

Interval Scheduling [KT 4.1]

Input: n intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$

Output: a maximum set of disjoint intervals.

For example, in  , the highlighted intervals form a maximum set of disjoint intervals. There are multiple optimal solutions in this example.

For this problem, we can imagine that we have a room, and there are people who like to book our room and they tell us the time interval that they need the room, and our objective is to choose a maximum subset of activities with no time conflicts.

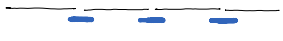
Generally speaking, greedy algorithms work by using simple and/or local rules to make decisions and commit on them. An analogy is to make maximum profit in short term.

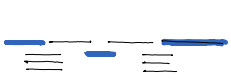
There are multiple natural greedy strategies for this problem including

- earliest starting time (choose the interval with $\min s_i$)
- earliest finishing time (choose the interval with $\min f_i$)
- shortest interval (choose the interval with $\min f_i - s_i$)
- minimum conflicts (choose the interval that overlaps with the minimum number of other intervals)

It turns that only one of these strategies would work.

Earliest starting time is the easiest to find counterexamples, because the interval with earliest starting time could be very long, e.g.  .

It is not difficult to find counterexamples for shortest intervals, e.g. 

It is a bit harder to find counterexamples for minimum conflicts, e.g. 

There are no counterexamples for earliest finishing time.

The intuition is that we leave maximum space for future intervals.

But how could we argue that this will always give an optimal solution (that there are no

solutions doing better)?

Algorithm

- Sort the intervals so that $f_1 \le f_2 \le \dots \le f_n$. Current solution $S = \phi$.
- For $1 \le i \le n$ do

  if interval i $[s_i, f_i]$ has no conflicts with other intervals in S, add i to S.
- Return S.

Correctness

The idea is to argue that any (optimal) solution would do no worse by using the interval with earliest finishing time.

Let $[s_{i_1}, f_{i_1}] < [s_{i_2}, f_{i_2}] \dots < [s_{i_k}, f_{i_k}]$ be the solution returned by the greedy algorithm.

Let $[s_{j_1}, f_{j_1}] < [s_{j_2}, f_{j_2}] \dots < [s_{j_\ell}, f_{j_\ell}]$ be an optimal solution with $\ell \ge k$.

Since $f_{i_1} \le f_{j_1} < s_{j_2}$ (the first inequality is because $i_1 = 1$ and is the interval with earliest finishing time, and the second inequality is because $[s_{j_1}, f_{j_1}]$ and $[s_{j_2}, f_{j_2}]$ are disjoint), so $[s_{i_1}, f_{i_1}] < [s_{j_2}, f_{j_2}] < \dots < [s_{j_\ell}, f_{j_\ell}]$ is still an optimal solution.

Thus we have the following claim, showing that it is no worse by choosing $[s_1, f_1]$.

Claim There exists an optimal solution with $[s_1, f_1]$ chosen.

We will use this argument inductively to prove the following lemma.

Lemma $[s_{i_1}, f_{i_1}], [s_{i_2}, f_{i_2}), \dots [s_{i_k}, f_{i_k}] [s_{j_{k+1}}, f_{j_{k+1}}] \dots [s_{j_\ell}, f_{j_\ell}]$ is an optimal solution with $f_{i_k} \le f_{j_k}$

Informally, the lemma says that the greedy solution always "stays ahead".

Before we prove the lemma, let's see how it implies that the greedy solution is optimal.

Suppose, by contradiction, that the greedy solution is not optimal, i.e. $\ell > k$.

Since $f_{i_k} \le f_{j_k} < s_{j_{k+1}}$, we see that the interval $[s_{j_{k+1}}, f_{j_{k+1}}]$ has no overlapping with the greedy solution, and it should have been added to the greedy solution by the greedy algorithm, a contradiction that the greedy algorithms only finds k disjoint intervals.

Proof of lemma The base case holds because of the previous claim.

Assume the claim is true for $c \ge 1$, and we are to prove the inductive step.

Since $[s_{i_1}, f_{i_1}] \dots [s_{i_c}, f_{i_c}] [s_{j_{c+1}}, f_{j_{c+1}}) \dots [s_{j_\ell}, f_{j_\ell}]$ is an optimal solution by the induction hypothesis with $f_{i_c} \le f_{j_c}$, we have $f_{i_{c+1}} \le f_{j_{c+1}}$ as the interval

$[s_{j_{c+1}}, f_{j_{c+1}}]$ has no overlapping with $[s_1, f_1], \ldots, [s_{i_c}, f_{i_c}]$ and the greedy algorithm chooses the next interval with earliest finishing time.

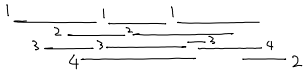
As $f_{i_{c+1}} \le f_{j_{c+1}}$, by replacing $[s_{j_{c+1}}, f_{j_{c+1}}]$ with $[s_{i_{c+1}}, f_{i_{c+1}}]$, the resulting solution $[s_{i_1}, f_{i_1}] \ldots [s_{i_{c+1}}, f_{i_{c+1}}] [s_{j_{c+2}}, f_{j_{c+2}}] \ldots [s_{j_k}, f_{j_k}]$ is feasible and optimal. □

Time complexity: It is easy to check that the greedy algorithm can be implemented in $O(n \log n)$ time.

Interval Coloring

Input: n intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$

Output: use the minimum number of colors to color the intervals, so that each interval gets one color and two overlapping intervals get two different colors.



(the numbers are the colors)

We can imagine this is the problem of using the minimum number of rooms (colors) to schedule all the activities (intervals).

One natural greedy algorithm is to use the previous algorithm to choose a maximum subset of disjoint intervals and use only one color for them, and repeat.

Find a counterexample for this algorithm!

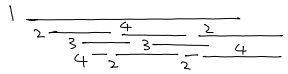
There is another greedy algorithm that will work.

Algorithm (interval coloring)

- Sort the intervals by starting time so that $s_1 < s_2 < \ldots < s_n$.
- for $1 \le i \le n$ do

  use the minimum available color c_i to color the interval i.

  (i.e. use the minimum number to color the interval i so that it doesn't conflict with the colors of the intervals that are already colored.)



Suppose the algorithm uses k colors.

To prove the correctness of the algorithm, we must prove that there are no other ways to color the intervals using at most $k-1$ colors.

How do we argue that ?

A nice and simple way is to show that when the algorithm uses k colors, there exists a time t such that it is contained in k intervals.

Since these k intervals are pairwise overlapping, we need at least k colors just to color them, and therefore a (k-1)-coloring doesn't exist.

Proof of correctness: Suppose the algorithm uses k colors.

Let interval ℓ be the first interval to use color k.

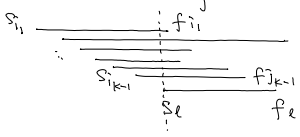
This implies that interval ℓ overlaps with intervals with colors $1, \ldots, k-1$.

Call them $[s_{i_1}, f_{i_1}], \ldots, [s_{i_{k-1}}, f_{i_{k-1}}]$.

As we sort the intervals with increasing order of starting time, we have $s_{i_j} \le s_\ell$ for all $1 \le j \le k-1$.

Since all these intervals overlap with $[s_\ell, f_\ell]$, we also have $s_\ell \le f_{i_j}$ $\forall$ $1 \le j \le k-1$.

Therefore, s_ℓ is a time contained in k intervals.



This implies that there is no k-1 coloring. □

Find a counterexample showing that using an arbitrary ordering of the intervals would not work.

Minimizing Total Completion Time

Input: n jobs, each requiring processing time p_i

Output: An ordering of the jobs to finish so as to minimize the total completion time.

(The completion time of a job is defined as the time when it is finished.)

For example, given four jobs with processing times 3, 4, 6, 7, and if we process the jobs in this order, then the completion times are 3, 7, 13, 20, and the total completion time is 43.

It is very intuitive that we should process the jobs in increasing order of processing time.

(Imagine we are in a supermarket with four customers with 3, 4, 6, 7 items.)

But how do we argue that there are no better solutions?

Here we use an "exchange argument" to show that the greedy solution is optimal.

(Again, we imagine that we have 1 item but the customer in front of us has 10 items.)

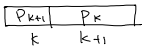
Proof of correctness

Consider a solution which is not sorted by non-decreasing processing time.

This implies that there is an "inversion pair": the i-th job and the j-th job with $i<j$ but $p_i > p_j$.

This implies that there exists $i \le k < j$ with $p_k > p_{k+1}$.

By swapping the jobs k and k+1, we will prove that the total completion time is smaller.

Note that all the completion times, except for k and k+1, are unchanged. (swap) 

Let C be the completion time of job k-1.

Before swapping, the completion times of job k and k+1 are $C + p_k$ and $C + p_k + p_{k+1}$ respectively. and the total of these two jobs are $2C + 2p_k + p_{k+1}$.

After swapping, the total of these two jobs are $2C + 2p_{k+1} + p_k$.

Since $p_{k+1} < p_k$, it is clear that the solution after swapping is better.

So, any ordering which is not sorted by non-decreasing processing times is not optimal. □

The exchange argument is quite nice and useful.
