CS 341 - Algorithms, Winter 2016. University of Waterloo

Lecture 8: Greedy algorithms

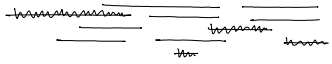
We start our discussions about greedy algorithms using some job scheduling problems as examples

---

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 could 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.

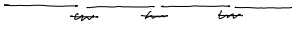
Generally speaking, greedy algorithms work by using some simple and/or local rules to make decisions and commit on them. (An analogy of making maximum profit in short term.)

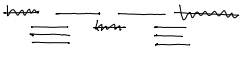
There are multiple natural "greedy" strategies including:

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

One will work and the other three will fail.

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 conflict with other intervals in S, add i to S.
- return S.

The idea is to argue that any (optimal) solution will 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 ($i_1 = 1$).

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), $[s_{i_1}, f_{i_1}] < [s_{j_2}, f_{j_2}] < \dots < [s_{j_\ell}, f_{j_\ell}]$ is still an optimal solution.

We have the following claim, showing that it is not worse by choosing $[s_1, f_1]$

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

This is basically the main argument.

We can use it inductively to prove the following.

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

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

Suppose, by contradiction, that the greedy solution is not an optimal solution, 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 algorithm could only find k disjoint intervals.

proof of claim The base case holds because of the previous claim.

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

Since $[s_{i_1}, f_{i_1}] \dots [s_{i_c}, f_{i_c}] [s_{j_{c+1}}, f_{j_{c+1}}] \dots [s_{j_k}, f_{j_k}]$ is an optimal solution by the induction hypothesis and $f_{i_c} \leq f_{j_c}$, we have $f_{i_{c+1}} \leq f_{j_{c+1}}$ as the interval $[s_{j_{c+1}}, f_{j_{c+1}}]$ has no overlapping with $[s_1, f_1], \dots, [s_{i_c}, f_{i_c}]$ and the greedy algorithm chooses the next interval with earliest finishing time.

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

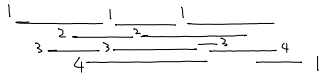
This finishes the proof of correctness.

Time complexity: We leave it as an exercise to check that the greedy algorithm can be implemented in $O(n \log n)$ time.

Interval Coloring

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

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



(the numbers are the colors)

You 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.

Exercise: Find a counterexample for this algorithm.

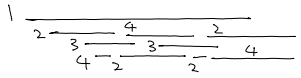
There is another very greedy algorithm that will work.

Algorithm (interval coloring)

- sort the intervals by starting time so that $s_1 < s_2 < \dots < s_n$.
- 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.)

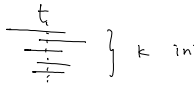


Suppose the algorithm use k colors.

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 simple way to do this to show that when the algorithm use 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.

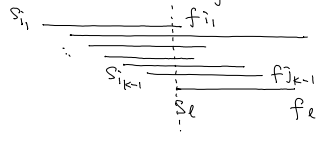
This implies that interval l overlaps with intervals with colors $1, \dots, k-1$.

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

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

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

Therefore, s_l is a time contained in k intervals.



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

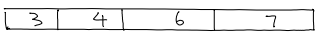
Exercise: Find a counterexample showing that using an arbitrary ordering of intervals will not work.

Minimize Total Completion Time

Input: n jobs, each requiring processing time p_i (assume p_i are positive integers)

Output: Find an ordering to process the jobs 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 time, 3, 4, 6, 7, if we process them in the ordering 

, then the completion times are 3, 7, 13, 20, and thus the total completion time is 43.

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

(Imagine this is a supermarket with four customers with 3, 4, 6, 7 items.)

But how do we argue that all other solutions won't do better?

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

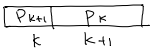
proof of correctness

Consider a solution which is not sorted by increasing 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 (a moment's though) that there exists $i \le k < j$ such that $p_k > p_{k+1}$.

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

All the completion times, except jobs k and k+1, are unchanged. (swap) 

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

In the old solution (before swapping), the completion time of job k and job k+1 are $C + p_k$ and $C + p_k + p_{k+1}$ respectivitely, and the total for these two jobs are $C + 2p_k + p_{k+1}$.

In the new solution (after swapping), the completion times of job k and job k+1 are $C + p_{k+1}$ and $C + p_{k+1} + p_k$ respectively, and the total for these two are $C + 2p_{k+1} + p_k$.

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

So, any solution which is not sorted by increasing processing time is not optimal. □

Minimizing Lateness [KT 4.2]

Input: n jobs, each with processing time p_i and deadline d_i

Output: find an ordering to schedule the jobs so as to minimize the maximum lateness

(i.e. Let C_i be the completion time of job i.

The lateness L_i is defined as $C_i - d_i$, and we want to min $\max_i L_i$).

It is not difficult to find a counterexample so that sorting by increasing processing time won't work.

We will show the other natural strategy of sorting by non-decreasing deadlines would work.

proof of correctness: We again use the exchanging argument.

Suppose there is a solution with ordering not sorted by non-decreasing deadlines.

This implies there exists k such that the k-th scheduled job has deadline larger than the $(k+1)$-th scheduled job, i.e. $d_k > d_{k+1}$.

Again, we consider a new solution by swapping the k-th and $(k+1)$-th job.

The lateness of every job

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

In the old solution, the completion times of job k and $k+1$ are $C + p_k$ and $C + p_k + p_{k+1}$.

The lateness of job k and $k+1$ are $L_k = C + p_k - d_k$ and $L_{k+1} = C + p_k + p_{k+1} - d_{k+1}$

In the new solution, the completion times of job k and $k+1$ are $C + p_{k+1}$ and $C + p_{k+1} + p_k$

The lateness of job k and $k+1$ are $L_k' = C + p_{k+1} - d_{k+1}$ and $L_{k+1}' = C + p_k + p_{k+1} - d_k$.

It is clear that $L_k' < L_{k+1}$.

Since $d_k > d_{k+1}$, it is also clear that $L_{k+1}' < L_{k+1}$.

Therefore, $\max\{L_k', L_{k+1}'\} < L_{k+1} \leq \max\{L_k, L_{k+1}\}$, and so the maximum lateness of the new solution won't increase.

By doing one swapping, we find a solution which is as good but with one fewer inversion pair.

Repeatedly swapping, we find a solution which is as good as the original solution, but with no inversion pair, i.e. Sorted by non-decreasing deadlines.

Starting from an optimal solution, this implies that the solution sorted by non-decreasing deadlines is as good, proving the optimality of the greedy solution. □

The exchange argument is quite nice and useful.
