CS 341: ALGORITHMS (S18) — LECTURE 7
MORE GREEDY ALGORITHMS

ERIC BLAIS

We saw in the last lecture how the greedy algorithm method can be used to solve the interval scheduling problem. In this lecture, we will examine a few other problems that can be solved by greedy algorithms and see how a common exchange method can be used to establish the correctness of these algorithms.

1. Minimizing lateness

There are a number of other scheduling problems for which the greedy algorithm is effective. Here's one that may be relevant with respect to juggling coursework for multiple classes at the same time:

Definition 7.1 (Minimal lateness problem)**.** An instance of the *minimal lateness problem* is a set of n tasks with processing times $p_1, \ldots, p_n$ and deadlines $d_1, \ldots, d_n$. A valid solution to the instance is an ordering of the tasks that minimizes the maximum lateness of any task when they are performed one at a time in that order. Formally, given a permutation π of the set $\{1, 2, \ldots, n\}$, the *lateness* of the kth task in this order is $L_k^{(\pi)} := \sum_{i:\pi(i)\le\pi(k)} p_i - d_k$ and π is a valid solution if $\max_{k\le n} L_k^{(\pi)}$ is minimal among all permutations.

For example, an input to the problem may be tasks with the following processing times and deadlines:

	1	2	3	4	5	6	7
p_i	4	2	4	3	1	4	6
d_i	5	9	1	3	3	4	10

If we process the tasks in the order above, we obtain lateness values

	1	2	3	4	5	6	7
L_i	-1	-3	9	10	11	14	14

so that the maximum lateness of this ordering is 14. For this instance, every ordering of the tasks has maximum lateness at least 14, so the order $1, 2, \ldots, n$ is a valid solution.

Let's explore how we can solve this problem using greedy algorithms. Here the natural way to break down the problem into individual decisions is to pick which task to do first, then which one to do second, etc. There are a number of different criteria we could use to decide which task to schedule next.

Shortest processing time first: Sort the tasks in order of increasing processing times.

Earliest deadline: Sort the tasks in order of increasing deadlines.

Smallest slack: At each step, pick the task with the smallest remaining time until its deadline: if the first j tasks already selected have total processing time t_j, pick the task that minimizes $d_j - t_j$ next.

Of those three options, only the earliest deadline criterion yields a greedy algorithm that solves the Minimal Lateness problem.[1]

```
Algorithm 1: GreedyLateness($p_1, \ldots, p_n, d_1, \ldots, d_n$)
$\pi \leftarrow$ a permutation of $\{1, \ldots, n\}$ for which $d_{\pi(1)} \leq d_{\pi(2)} \leq \cdots \leq d_{\pi(n)}$;
return $\pi$;
```

The permutation π can be computed in time $\Theta(n \log n)$, and this is also the time complexity of the algorithm. The more challenging aspect of the analysis of this algorithm is its proof of correctness. We again use an exchange argument. We also use the fact that we can sort an out-of-order sequence by swapping pairs of consecutive elements.

Fact 1 (Bubble sort)**.** *For π any ordering of $1, 2, \ldots, n$, if we repeatedly*

(1) *Find a pair of elements $i < j$ where j is right before i in π;*
(2) *Update π by swapping elements i and j;*

then after at most $\Theta(n^2)$ swaps, we end up with the sorted ordering $1\,2\,3 \cdots n$.

For example, starting with the ordering $1\,5\,3\,4\,2$, the sorting algorithm described above yields the following sequence of orderings:

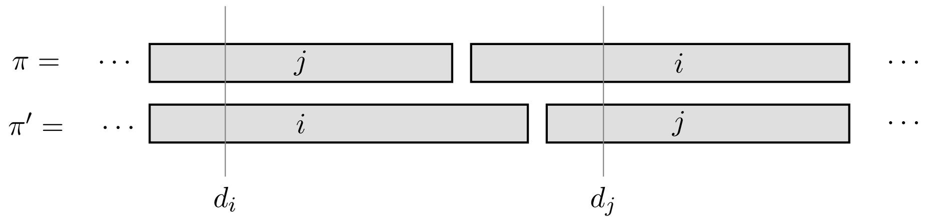
$$1\,\mathbf{5}\,\mathbf{3}\,4\,2$$
$$1\,3\,\mathbf{5}\,\mathbf{4}\,2$$
$$1\,3\,4\,\mathbf{5}\,\mathbf{2}$$
$$1\,3\,\mathbf{4}\,\mathbf{2}\,5$$
$$1\,\mathbf{3}\,\mathbf{2}\,4\,5$$
$$1\,2\,3\,4\,5$$

We are now ready to prove the correctness of the GreedyLateness algorithm.

Theorem 7.2. *The* GreedyLateness *algorithm solves the Maximal Lateness problem.*

Proof. For simplicity, let us reorder the tasks by increasing deadline so that the greedy algorithm performs them in order (i.e., 1, then 2, then 3, etc.) We want to prove that this order is a valid solution.

Consider any other ordering π. Then in that order there must be two consecutive tasks i, j with $d_i \leq d_j$ but j performed right before i. Swap those two tasks to obtain a new ordering π'.



[1]Exercise: Prove that statement!

Then every task except i and j have the same lateness in π and in π'. The lateness of i satisfies

$$L_i^{(\pi')} \leq L_i^{(\pi)}$$

since we do task i earlier in π'. And the lateness of j satisfies

$$L_j^{(\pi')} \leq L_i^{(\pi)}$$

because $d_i \leq d_j$. Therefore, the maximum lateness of π' is at most that of π. This means that we can continue swapping in this way until we obtain the ordering $1\,2 \cdots n$ of the greedy algorithm, and at every step along the way we never increase the maximum lateness so that the algorithm's solution is valid. □

2. Fractional knapsack

Now a completely different example.

Definition 7.3 (Fractional knapsack)**.** An instance of the *fractional knapsack problem* is a set of n items that have positive weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$, as well as a maximum weight capacity W of the knapsack. A valid solution to the problem is a set of amounts $x_1, \ldots, x_n$ of each item that you put in your backpack that maximizes the total value $\sum_{i=1}^n \frac{x_i}{w_i} v_i$ among all possible sets of amounts where for each item i we have $0 \leq x_i \leq w_i$ and the total weight taken is $\sum_{i=1}^n x_i \leq W$.

For example, an instance may have three items with weights and values

	1	2	3
w_i	4	3	3
v_i	12	7	6

and a total knapsack capacity weight $W = 6$.

There is a natural greedy algorithm for this problem: sort the items by decreasing relative value v_i/w_i, then consider each element in turn and put as much of it in the knapsack as can fit.

```
Algorithm 2: GreedyKnapsack($w_1, \ldots, w_n, v_1, \ldots, v_n, W$)
Order the items by decreasing value of $v_i/w_i$;
for $i = 1, \ldots, n$ do
    $x_i \leftarrow \min\{W, w_i\}$;
    $W \leftarrow W - x_i$;
return $x_1, \ldots, x_n$;
```

Theorem 7.4. *The* GreedyKnapsack *algorithm solves the fractional knapsack problem.*

Proof. Let $y_1, \ldots, y_n$ be a valid solution to the fractional knapsack problem, using the ordering defined by the algorithm where v_i/w_i is decreasing. And let $x_1, \ldots, x_n$ be the solution returned by the GreedyKnapsack algorithm. We prove that $x_1, \ldots, x_n$ is a valid solution by induction on the number of indices $i \leq n$ for which $x_i \neq y_i$.

In the base case, when $x_i = y_i$ for each $i = 1, 2, \ldots, n$, then $x_1, \ldots, x_n$ is the same solution as $y_1, \ldots, y_n$ so it is a valid solution.

For the induction step, let $m = |\{i \leq n : x_i \neq y_i\}| \geq 1$ be the number of differences in the solution. By the induction hypothesis, if there is a valid solution $y'_1, \ldots, y'_n$ with

$|\{i \leq n : x_i \neq y'_i\}| < m$, then $x_1, \ldots, x_n$ is also a valid solution. Our goal is to now use an exchange argument to show that such a solution $y'_1, \ldots, y'_n$ exists.

Let $k \leq n$ be the smallest index where $x_k \neq y_k$. Then it must be that $x_k > y_k$ since GreedyKnapsack maximizes the value of x_k. And since $\sum x_i = \sum y_i$, there must be an index $\ell > k$ for which $y_\ell > x_\ell$. Let's exchange a δ amount of weight of element ℓ for element k to obtain the solution y' where

$$y'_k = y_k + \delta \qquad \text{and} \qquad y'_\ell = y_\ell - \delta.$$

Choose $\delta = \min\{x_k - y_k, y_\ell - x_\ell\}$ so that $y'_k = x_k$ or $y'_\ell = x_\ell$ and, therefore, $|\{i \leq n : x_i \neq y'_i\}| < m$. To complete the proof, we must show that $y'_1, \ldots, y'_n$ is a valid solution. The difference in the total value of the solutions y' and y satisfies

$$\delta(v_k/w_k) - \delta(v_\ell/w_\ell) = \delta(v_k/w_k - v_\ell/w_\ell).$$

The ordering of the elements guarantees that $v_k/w_k \geq v_\ell/w_\ell$ so that the total value of y' is at least as large as that of y and, therefore, y' is a valid solution. □

What if we now consider the variant of the problem where we are only allowed to choose $x_i \in \{0, w_i\}$? (I.e., where the items are indivisible.) Does the greedy algorithm above still work? You should convince yourself that it is no longer correct—and you should be able to see how the exchange argument we used above fails in this situation! We will revisit this version of the knapsack problem in the next section, when we see how the *dynamic programming* algorithm design technique can be used to solve it efficiently when W is not too large. (And we will see the problem again in the NP-completeness section, when we will see why we shouldn't expect to find an efficient algorithm for this problem in general.)

3. Bonus: Another proof of correctness for maximum lateness

The argument we developed earlier to establish the correctness of the GreedyLateness algorithm is not the only possible argument we could use. There is another exchange argument where we use a proof by induction on the number of *inversions* in an ordering of the tasks.

Definition 7.5 (Inversions)**.** The *number of inversions* between two permutations π and σ of $\{1, 2, \ldots, n\}$ is

$$\text{inv}(\pi, \sigma) = |\{1 \leq i \neq j \leq n : \pi(i) < \pi(j) \wedge \sigma(i) > \sigma(j)\}|,$$

the total number of pairs of elements in the sequence that have different relative order in π and σ.

We use two basic facts about the number of inversions.

Fact 2. *Two permutations π and σ satisfy* $\text{inv}(\pi, \sigma) = 0$ *if and only if* $\pi = \sigma$.

Fact 3. *When two permutations π and σ on $\{1, \ldots, n\}$ satisfy* $\text{inv}(\pi, \sigma) \geq 1$*, then there must exist an index $k < n$ such that $\pi(k) > \pi(k+1)$ and $\sigma(k) < \sigma(k+1)$.*[2]

It is also convenient for this proof to change the definitions slightly. For a given instance of the problem, redefine the *lateness* of the kth task in this order to be $L_k^{(\pi)} := \sum_{i \leq k} p_{\pi(i)} - d_{\pi(k)}$. (I.e., the difference is that with this notation $L_k^{(\pi)}$ is the lateness of the kth task performed in the order defined by k, not the original task k.)

[2]Exercise: And prove this statement as well!

Theorem 7.6. *The* GreedyLateness *algorithm solves the Maximal Lateness problem.*

Proof. Let σ be an ordering of $\{1, 2, \ldots, n\}$ that is a valid solution to the maximal lateness problem, and let π be the ordering returned by the GreedyLateness algorithm. We prove that π is a valid solution by induction on the number of *inversions* between π and σ.

In the base case, when $\mathrm{inv}(\pi, \sigma) = 0$ then $\pi = \sigma$ so π is a valid solution.

For the induction step, let us now consider a valid solution σ with $\mathrm{inv}(\pi, \sigma) = m \geq 1$ inversions and the induction hypothesis is that if we have a valid solution σ' with $\mathrm{inv}(\pi, \sigma') \leq m - 1$ inversions, then π is also a valid solution. Let $k < n$ be an index for which $\pi(k) > \pi(k + 1)$ and $\sigma(k) < \sigma(k + 1)$. The fact that $\pi(k) > \pi(k + 1)$ implies that we must have $d_{\sigma(k)} \geq d_{\sigma(k+1)}$. Let σ' be the permutation obtained by exchanging the values $\sigma'(k + 1) = \sigma(k)$ and $\sigma'(k) = \sigma(k + 1)$ and leaving the rest of the values $\sigma'(i) = \sigma(i)$ as in σ when $i \notin \{k, k + 1\}$.

To complete the proof, we want to show that the maximal lateness of σ' is no larger than the maximum lateness of σ. First, we observe that for every $i < k$ and for every $i > k + 1$, we have $L_i^{(\sigma')} = L_i^{(\sigma)}$. So the only latenesses that are different in σ and σ' are for $i = k$ or $i = k + 1$. Define $C = \sum_{i<k} p_{\sigma(i)}$. Then $C = \sum_{i<k} p_{\sigma'(i)}$ as well, so

$$\begin{aligned} L_k^{(\sigma')} &= C + p_{\sigma'(k)} - d_{\sigma'(k)} \\ &= C + p_{\sigma(k+1)} - d_{\sigma(k+1)} \end{aligned}$$

and

$$\begin{aligned} L_{k+1}^{(\sigma')} &= C + p_{\sigma(k)} + p_{\sigma(k+1)} - d_{\sigma(k+1)} \\ &= C + p_{\sigma(k)} + p_{\sigma(k+1)} - d_{\sigma(k)}. \end{aligned}$$

Meanwhile,

$$L_{k+1}^{(\sigma)} = C + p_{\sigma(k)} + p_{\sigma(k+1)} - d_{\sigma(k+1)}.$$

Since $p_{\sigma(k+1)} \geq 0$, then $L_k^{(\sigma')} \leq L_{k+1}^{(\sigma)}$. And since $d_{\sigma(k)} > d_{\sigma(k+1)}$, then $L_{k+1}^{(\sigma')} \leq L_{k+1}^{(\sigma)}$ also. Therefore,

$$\max\left\{L_k^{(\sigma')}, L_{k+1}^{(\sigma')}\right\} \leq L_{k+1}^{(\sigma)} \leq \max\left\{L_k^{(\sigma)}, L_{k+1}^{(\sigma)}\right\}$$

and σ' must be a valid solution as well. $\square$