Illinois Institute of Technology
Department of Computer Science

Lecture 18: March 27, 2019

CS 330 Discrete Structures
Spring Semester, 2019

Greedy Algorithms

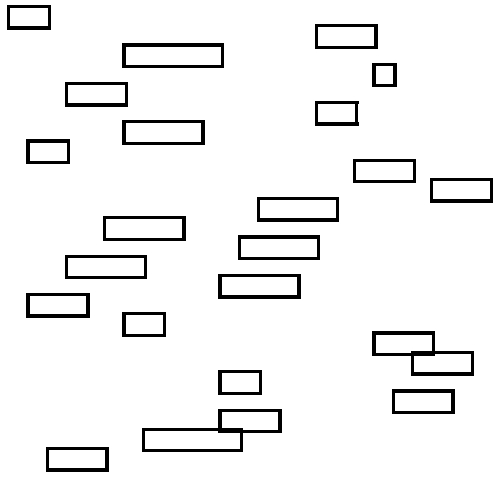
We have discussed Divide-and-Conquer algorithms as a general paradigm; now we discuss another paradigm, **Greedy Algorithms**. Greedy Algorithms are appropriate for many types of problems. However, there are still problems (such as the Traveling Salesman problem that in which a greedy algorithm is not optimal. We will now look at examples where the greedy algorithm works well.

1 Activity Selection

In this sub-section we will study scheduling. Here's the problem:

You are given a list of programs to run on a single processor; each program has a start time and finish time when it should run. However, the processor can only run one program at any given moment. Your task is to schedule the maximum number of programs to run on the processor.

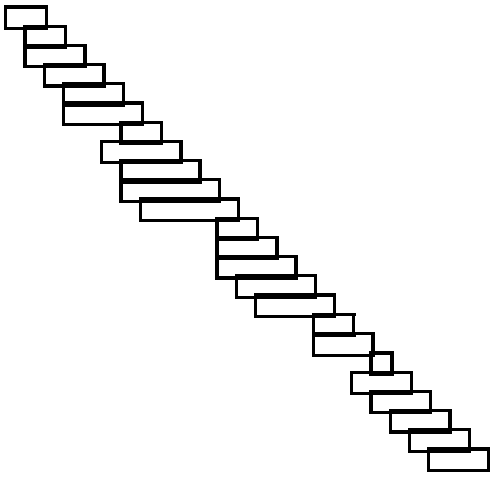
This problem can also be cast as an activity-scheduling problem, where activities take the place of programs. For example, you could be given the following list of activities:



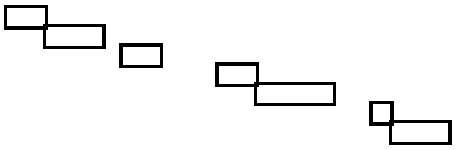
This actually looks too simplified to have any practical use in scheduling jobs for real computers. Besides, in real life it's hard to give exact start and finish time for jobs. But this problem can happen in real life. Suppose a guy with bad taste surprisingly won a Tech News award for guessing Oscar winners and got a one-day pass at Beverly. The cinema has a lot of halls, each of them with different schedules. The guy, of course, can only view one at a time and he only watches complete movies. The goal of this guy is to view as many free movies as possible even if he need to view the same one twice. This is exactly our simplified job scheduling problem with the movies as activities and this greedy guy as the computer.

There are many ways to do optimally schedule these activities. In the exhaustive method, we could just examine every subset of the elements and see which is the best one. There are 2^n ways to make a subset of a set of size n, so our algorithm will require exponential time.

If we allow for some preprocessing, we could significantly improve this algorithm. Specifically, we can first sort the items to be scheduled by their finish time:



In this algorithm, we start at the top of our sorted list and include the first activity. We then go through the list in order including activities whose start time is after (or the same as) the last added activity. This is called the `GREEDY ACTIVITY SELECTOR`, because we are picking the next activity to add based on a "greedy" criterion, that is the activity that looks best at the time we have to choose. For this particular set of events, we will end up with the following selection of activities:



This algorithm visits each item to be scheduled once, therefore the selection part requires time proportional to n. Remember that we first sorted the list, which required $O(n \log n)$ time, so the entire algorithm has a complexity of $O(n + n \log n)$ which is $O(n \lg n)$. This is much better than 2^n, but there is a potential problem. The exhaustive algorithm guaranteed us an optimal scheduling; does this algorithm give us an optimal solution. In other words, does making locally optimal ("greedy") decisions give a globally optimal solution in this case?

The answer is "yes" and we prove the optimality of our greedy algorithm by comparing the set selected with the greedy algorithm (called A) with a true optimal set (called B). If A is not optimal, there must be a place where the greedy algorithm selected a non-globally-optimal choice. If a non-optimal choice was made, there must be a first non-optimal choice. Let's see where that first non-optimal choice was.

Proof of the Optimality of the Greedy Algorithm

Let $A = \{a_{x_0}, a_{x_1}, a_{x_2}, a_{x_3}, \cdots\}$ be the greedily chosen set of activities. Suppose that after picking $j-1$ activities correctly, the greedy algorithm makes a mistake. It picks an activity a_{x_j} that could not possibly lead to an optimal solution. Let us say that a correct choice of activities was b_{x_j} that would lead to an optimal solution $B = \{a_{x_0}, a_{x_1}, a_{x_2}, a_{x_3}, \cdots, a_{x_{j-1}}, b_{x_j}, b_{x_{j+1}}, \cdots\}$.

Clearly, a_{x_j} must end before b_{x_j} or else our greedy selection would have picked b_{x_j}. Thus, switching a_{x_j} and b_{x_j} in the optimal solution B (to get a selection B') must still give a valid activity selection (think about this: remember that a_{x_j} was an acceptable choice for activities after $j-1$ iterations). However, B' contains the same number of elements as B and is, hence, optimal. Thus, the greedy choice does indeed provide the possibility for an optimal solution. Since each choice of activities by the greedy strategy allows for an optimal solution, and we continue choosing activities until exhaustion, the greedy activity selection must be optimal.

2 Greedy algorithms and suboptimal solutions

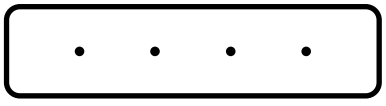
Greedy algorithms tend to be simple and efficient, as the locally optimal solution to a problem is usually easy to find. Unfortunately, greedy algorithms do not always yield optimal solutions.

Consider a set of points on the plane. We would like to connect each point to exactly one other point in such a way that the cost, the sum of the lengths of the edges, is minimized.

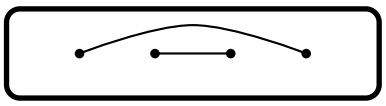
A brute force approach to this problem is to generate each possible matching of points and to find the corresponding costs. The matching with the minimum cost is thus the solution to the problem. While this approach will inarguably produce correct results, it requires exponential time, so we look for a more efficient algorithm.

One simple algorithm that comes to mind is a greedy algorithm. Find the two nearest points that are not yet connected (to other points) and connect them. Repeat this process exhaustively. In the case of a tie, select a pair of points arbitrarily. How efficient is this algorithm? If there are n points, it will take time $\binom{n}{2} + \binom{n-2}{2} + \binom{n-4}{2} + \cdots = \Theta(n^3)$.

Before declaring success, we must ask if our algorithm is correct: does it, in fact, give optimal results? Unfortunately it does not. Consider four points equally spaced along a line:

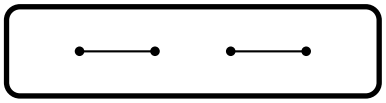


In the first iteration, there are three optimal pairs of points, so the algorithm chooses a pair arbitrarily. Say it chooses the two points in the middle:[1]

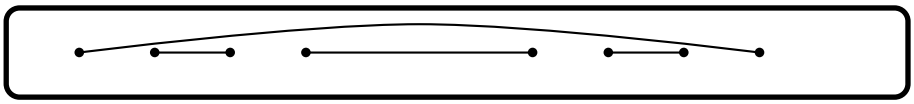


[1] Alternatively, let the points be separated in turn by distances 1, $1-\epsilon$, and 1, where ϵ is some small positive number. Then the algorithm is forced to select the two points in the middle.

This is clearly not the optimal solution, however. The optimal solution connects the two leftmost points and the two rightmost points instead:



If we separate two of the above case by three positions, we have an even worse case:



Again, the optimal solution does much better:

We can repeat this process to derive even worse results.

Consider the optimal solution to the problem with $N = 2^n$ points. The minimum cost is $\text{OPTIMUM}_n = 2^{n-1}$, where $\text{OPTIMUM}_1 = 1$.

Compare this to the solution given by the greedy algorithm. Let L_n be the length from the leftmost point to the rightmost point if there are 2^n points. This leads to the recurrence:

$$\begin{aligned}
\text{GREEDY}_n &= 2\text{GREEDY}_{n-1} + L_n - 2L_{n-1} + L_{n-1} \\
&= 2\text{GREEDY}_{n-1} + L_n - L_{n-1} \\
&= 2\text{GREEDY}_{n-1} + 3^{n-1} - 3^{n-2}.
\end{aligned}$$

This is annihilated by $(\mathbf{E} - 2)(\mathbf{E} - 3)$, leading to the solution:

$$\text{GREEDY}_n = 2 \cdot 3^{n-1} - 2^{n-1}.$$

The error in the greedy algorithm relative to the optimal solution, then, is:

$$\begin{aligned}
\frac{\text{GREEDY}_n}{\text{OPTIMUM}_n} &= \frac{2 \cdot 3^{n-1} - 2^{n-1}}{2^{n-1}} \\
&= 2 \cdot \left(\frac{3}{2}\right)^{n-1} - 1 \\
&= \frac{4}{3}\left(\frac{3}{2}\right)^{n} - 1 \\
&= \frac{4}{3}N^{\lg \frac{3}{2}} - 1 \\
&\approx O(\sqrt{N}).
\end{aligned}$$

Not only is the solution given by the greedy algorithm not optimal, as the size of the problem grows, the error grows as well. Clearly for certain problems a greedy approach is inappropriate.