1 More Perspectives on Greedy

- We’re now going to look at some more examples of greedy algorithms.
- There are other strategies besides “greedy stays ahead” to prove optimality, which we’ll see.
- First, let’s continue with our investigation of scheduling with a new variation.
- Wash U. needs to schedule final exams for $n$ classes.
- Each exam $i$ occupies a time interval $[s_i, e_i]$.
- The university has a large number of rooms in which to schedule exams; only one exam may use a room at a time.
- **Problem**: schedule all $n$ exams using as few rooms as possible.
- (This problem is sometimes called “interval graph coloring.”)

Before we try to solve the problem, how well could we possibly do?

- Is there a natural lower bound on the number of rooms needed for a solution?
- Sure – if $k$ exams are in progress simultaneously at some point in time, we need to use at least $k$ rooms.
- In general, let the depth $d(R)$ of a problem instance $R$ be the maximum number of simultaneous exams, i.e.
  \[
  d(R) = \max_{t} |\{r \in R \mid r \text{ overlaps } t\}| .
  \]
- No algorithm can schedule $R$ using fewer than $d(R)$ rooms.
- If an algorithm uses only $d(R)$ rooms, its solution is definitely optimal.

So, can we find an algorithm that achieves the bound $d(R)$?

- Assume rooms are numbered in some total order 1, 2, 3...
- Sort the exams by starting time.
• For each exam \(i\) in this order...

• Let \(C_i\) be the list of all rooms assigned to exams \(j < i\) s.t. \(j\) conflicts with \(i\).

• Assign \(i\) to the lowest-numbered room not in \(C_i\).

• **Claim:** this algorithm produces a feasible schedule for any problem instance \(R\) with optimal depth \(d(R)\).

• **Pf:** first, observe that the algorithm never assigns two conflicting intervals \(j < i\) to the same room, because \(j\) appears in \(C_i\) by definition. Hence, the solution is feasible.

• Second, suppose the algorithm assigns an interval \(i\) to room \(k\). Then \(C_i\) must contain conflicting intervals for all \(k' < k\).

• All \(k - 1\) of these conflicting intervals started before \(i\), so they all overlap time \(s_i\).

• Conclude that \(d(R) \geq k\).

• Hence, the algorithm always returns a solution that uses \(\leq d(R)\) rooms, and so it is optimal. QED

Our algorithm is in some sense “greedy” – it blindly assigns each exam to the lowest available room. Optimality is proved by showing a *lower* bound on the value of the solution, regardless of how it is obtained, and then showing that the greedy algorithm achieves it.

## 2 Network Design with Minimum Spanning Trees

Let’s talk about a well-studied problem in network building that admits greedy algorithms.

• We are given a collection of cities \(v_1 \ldots v_n\).

• We can connect pairs of cities by railroad lines.

• It costs \(c(v_i, v_j)\) to lay track between cities \(v_i\) and \(v_j\).

• We seek to build enough railroad lines so that a passenger starting in any city can travel to any other city in one or more hops.

• To save money, we want to build a set \(T\) of railroad lines of minimum total cost \(\sum_{(v_i, v_j) \in T} c(v_i, v_j)\).

• More abstractly, we are given a graph over a collection \(V\) of vertices.

• We wish to find a set \(T \subseteq V \times V\) of (undirected) edges between vertices.

• For any two vertices \(v_i, v_j\), \(T\) must contain a path of edges connecting \(v_i\) to \(v_j\).

• We say that \(T\) *spans* the vertex set \(V\).

• We wish to find a spanning edge set \(T\) for \(V\) of minimum total cost.

What can we say about the edge set \(T\)?
• Suppose that every edge has $c(v_i, v_j) > 0$.

• Can $T$ contain a cycle, that is, a path of one or more edges from a vertex back to itself?

• **Claim**: A spanning edge set of minimum cost cannot contain a cycle.

• **Pf**: Suppose $T$ contains a cycle $U$ of vertices $u_1, u_2, \ldots, u_{\ell-1}, u_\ell$, with $u_\ell = u_1$.

• We claim that we can remove any one edge from $U$ and still have a set of edges that spans $V$.

• Indeed, suppose we remove edge $(u_i, u_{i+1})$.

• Any path through $V$ that uses this edge can instead follow the alternate path $u_i, u_{i-1}, \ldots, u_1 = u_\ell, u_{\ell-1} \ldots u_{i+1}$.

• But the removed edge has non-zero cost, so the resulting spanning edge set has lower cost than $T$, which we assumed to have minimum cost! →←

• Conclude that $T$ cannot have any cycles. QED

• Because $T$ is acyclic, it is a tree on $V$.

• In particular, we call $T$ a *minimum spanning tree* (MST) for $V$.

3 Efficient Minimum Spanning Tree Construction

Let’s look at a couple of greedy algorithms for building MSTs.

• The following algorithm is due to Joseph Kruskal (1956).

• Initially, let $T = \emptyset$.

• While $T$ does not span $V$...

• Let $e_i$ be the lowest-cost edge such that $T \cup \{e_i\}$ does not contain a cycle.

• Add $e_i$ to $T$ and continue.

• (A spanning tree on $n$ vertices contains exactly $n - 1$ edges, so this algorithm terminates after $n - 1$ steps.)

• A closely related algorithm is was published by Prim and Dijkstra (1957), though Jarnik (1930) discovered it first.

• Initially, let $T = \emptyset$.

• While $T$ does not span $V$...
• Let $e_i$ be the lowest-cost edge connecting some vertex $v$ touched by $T$ to some vertex $u$ not touched by $T$.
• Add $e_i$ to $T$ and continue.
• (Prim’s algo ensures that at any point in the algorithm, $T$ is always a tree on some subset of $V$. Kruskal’s algorithm maintains a forest that only eventually becomes a single tree.)

These algorithms are both greedy. Why are they correct?
• First, neither algorithm’s greedy choice ever connects two vertices that are already connected by a path in $T$.
• Hence, the final $T$ from each algorithm is a tree.
• Moreover, the algorithms output a connected graph $T$ that spans $V$ (obvious for Prim, mostly so for Kruskal).
• The key question is, why are the algorithms’ spanning trees minimum?
• We’re going to prove a key property of the two algorithms now, and then turn it into a proof of optimality later.
• For simplicity, let’s assume that all the edges in the graph have distinct costs – no ties!
• **Lemma**: let $S$ be a nonempty, proper subset of $V$ (sometimes called a “cut” of $V$).
• Let $(v, u)$ be the edge of minimum cost connecting some $v \in S$ to some $u \in V - S$.
• Then $(v, u)$ must be part of any MST for $V$.
• (Note that Prim’s and Kruskal’s algorithms each form a cut of $V$ at each step, then pick the lowest-cost edge that crosses from one side of the cut to the other.)
• **Pf**: Let $T$ be a spanning tree for $V$, and let $S$ be a cut of $V$.
• Suppose $T$ does not contain the edge of minimum cost $(v, u)$ connecting $S$ with $V - S$.
• Because $T$ spans $V$, it must contain some other path $p$ from $u$ to $v$.
• Now $v \in S$ and $u \in V - S$, so at some point, path $p$ crosses from $S$ to $V - S$.
• In particular, suppose it crosses at edge $(v', u')$: 
• Let $T' = T - \{(v', u')\} \cup \{(v, u)\}$.

• (That is, we throw out edge $(v', u')$ and replace it with $(v, u)$.)

• We claim that $T'$ still spans $V$.

• Indeed, any path that uses edge $(v', u')$ can be rerouted through $(v, u)$ instead, so $T'$ still spans $V$.

• Moreover, $T'$ is still a tree.

• Adding $(v, u)$ to $T$ can create at most a single cycle. If $T \cup \{(v, u)\}$ had two cycles, both must pass through $(v, u)$ (since $T$ was a tree).

• But then the union of these two cycles, minus $(v, u)$, form a cycle in $T$!

• The single cycle in $T \cup \{(v, u)\}$ can be broken by removing any one of its edges, including $(v', u')$.

• **One more claim:** $T'$ has lower total cost than $T$.

• This follows immediately because $(v, u)$ is the unique lowest-cost edge connecting $S$ to $V - S$.

• But then $T$ cannot be an MST!

• Conclude that no spanning tree without $(v, u)$ can be minimum, and so $(v, u)$ is in every MST of $V$. QED

What did we do here?

• Given any feasible solution $T$ (spanning tree) for a graph, and a cut $S$, we can *exchange* some edge $(v', u') \in T$ crossing from $S$ to $V - S$ for our greedily chosen lowest-cost crossing edge $(v, u)$.

• Doing so gives us a new feasible solution $T'$ that is better than the original $T$.

• Therefore, selecting $(v, u)$ must be consistent with some optimal feasible solution – any solution that doesn’t use it is not optimal.

• We’ll see that the “exchange argument” technique is the basis of a quite general strategy for finding proofs of optimality for greedy algorithms.

• More next time...