Lecture 5**:** The Greedy Algorithm

Date: February 4, 2026 **Scribe:** Lily Renneker

For today's lecture, we take a brief but important detour into the world of algorithms. While our recent focus has been on the structural properties of objects, particularly trees, and on various combinatorial proof techniques, understanding the algorithmic side of combinatorics is equally essential.

1 The Optimization Problem

Let $G = (V, E)$ be a **connected** graph:

- There is a cost function $c : E \to \mathbb{R}_{>0}$.
- We denote the cost of $e \in E$ as c_e.
- For a subset $T \subseteq E$, let $C(T) = \sum_{e \in T} c_e$

Problem: Find a minimum cost **connected** subgraph of G. More formally, find:

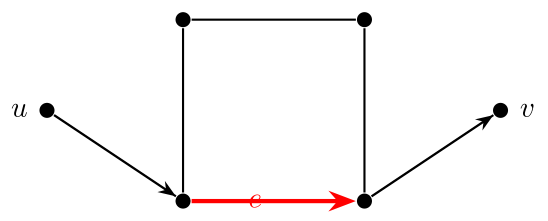
$$T^* \in \underset{\substack{T \subseteq E \\ T \text{ is connected}}}{\arg\min} \; C(T)$$

Goal: The goal of this lecture is to highlight a few different combinatorial arguments:

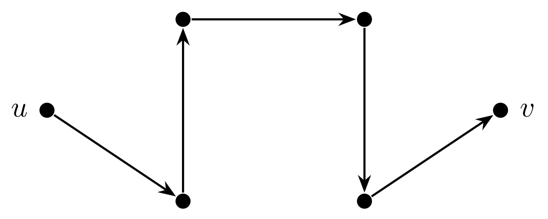
1. Algorithmic invariant
2. Characterizing optimal solutions via exchange arguments

Lemma 1.1. *Let $G = (V, E)$ and $C : E \to \mathbb{R}_{>0}$. Then, a minimum cost $T^* \subseteq E$ that is connected is a tree*

Proof. Supposed T^* contains a cycle C. Pick any $e \in C$. Note that $(V, T^* - e)$ is still connected. That is, any u, v-path in T^* that uses e can be rerouted:



A connected graph containing a cycle; the edge e is highlighted.



the edge e is removed, cycle is rerouted.

But, $C(T^* - e) < C(T^*)$ and we have found a contradiction. Thus, T^* can be limited to trees. $\square$

<u>Recall:</u> $|\{T \subseteq E : T \text{ is a tree}\}| \leq n^{n-2}$ cam be incredibly large

- Combinatorial optimization is about navigating search spaces with incredibly large (but finite) size
- How can we "efficiently" find a solution when search space is as large as n^{n-2}?

Before introducing the greedy algorithms, we ask the question, **what does it mean to be a greedy algorithm?** The main idea is to what looks "best" here and now, and hope for the best in the long run.

- i.e. we have a problem, and make decisions sequentially based on the decision at the moment, without consideration of quality of the final solution.

2 Algorithms

To exemplify the optimization problem, we will consider two different greedy algorithms:

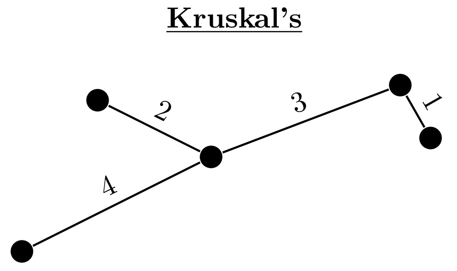
Algorithm 1 Kruskal's Algorithm

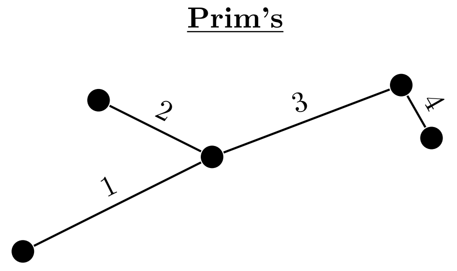
```
1: Sort $E$ in increasing order of cost $c_e$
2: $T \leftarrow \emptyset$
3: for each $e \in E$ do
4:     if $T + e$ has no cycle then
5:         $T \leftarrow T + e$
6:     end if
7: end for
8: return $T$
```

Algorithm 2 Prim's Algorithm

```
1: Pick arbitrary $s \in V$ as a root
2: $T \leftarrow \emptyset$
3: $S = \{s\}$
4: while $S \neq V$ do
5:     Let $e^* \in \arg\min_{e \in \delta(S)} c_e$
6:     $T \leftarrow T + e$
7:     $S \leftarrow S \cup e$
8: end while
9: return $T$
```

Here, for any $\emptyset \neq S \subset V$, we let $\delta(S) := \{e \in E : |e \cap S| = 1\}$. To illustrate the execution of these algorithms, consider the following instance:





The cost, in this case, is the euclidean distance between nodes. For these two algorithms, the same tree is recovered, but the order in which you return the edges is different (as denoted by the numbers next to the edges).

Theorem 2.1. *Prim's and Kruskal's algorithms are **optimal**, in the sense that they return a minimum cost spanning tree*

Some other ideas of "optimal" for algorithms include (although not covered in this course):

- time efficiency (time complexity)
- storage/memory space

Before we prove Theorem 2.1, we will first define a structural property of optimal solutions:

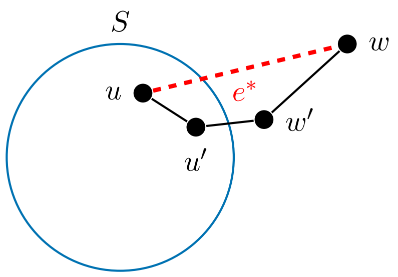
Lemma 2.2. *Suppose the costs c_e for $e \in E$ are pairwise distinct. Let $\emptyset \neq S \subset V$ and let*

$$e^* \in \arg\min_{e \in \delta(S)} c_e.$$

Then, e^* belongs to every minimum spanning tree.

Proof. Fix any such S and let T be some tree such that $e^* \notin T$. We need to show that $C(T) > C(T^*)$, where T^* is a minimum spanning tree. Let $e^* = \{u, w\}$. Since T is connected (by definition of tree), there exists a unique $\{u, w\}$-path in T. Without loss of generality, suppose $u \in S$ and $w \notin S$. Let u' be the last node in the path from u to w such that $u' \in S$. Let w' be the next node along this path, and note that $w' \notin S$ (by our choice of u').

The following figure illustrates the situation.



Let $T' = T - e' + e^*$ where $e' = \{u', w'\}$. We note that:

1. T' is connected (this follows as in the proof of Lemma 1.1, in which paths are re-routed.)
2. T' is acyclic (since T was acyclic to begin with, and the only cycle created by adding e^* is broken by removing e').
3. $C(T') < C(T)$

Therefore, if $e^* \notin T$, then T cannot be an optimal solution. $\square$

Now, we will prove Theorem 2.1.

Proof. We want to show that the algorithms maintain the invariant that only edges meeting property:

$$e^* \in \arg\min_{e \in \delta(S)} c_e$$

are picked, and that a spanning tree is returned.

First we will prove thereom 2.1 for Kruskal's algorithm. Suppose the algorithm adds some edge $e = \{v, w\}$. Let S be the component of v right before e is brought into the solution. That is, S is some connected component containing v. Thus, $w \notin S$ since e does not create a cycle (by the definition of the algorithm).

This means that $\emptyset \neq S \subset V$. Also, e is the first edge in $\delta(S)$ to be considered for bringing into solution T, since any previous edges in $\delta(S)$ could have been added without creating a cycle. This means that,

$$e \in \arg\min_{e \in \delta(S)}, c_e$$

since we evaluate edges from $\delta(S)$ in increasing order of cost.

Finally, algorithm returns T that is acyclic by the definition of the algorithm and connected, for otherwise, there is an edge that can be added.

Next, proving Prim's algorithm, we see that the choice of edges is precisely:

$$e \in \arg\min_{e \in \delta(S)} c_e$$

by the definition of the algorithm and thus the proof is more immediate. $\square$

Observations:

1. These algorithms also work if the objective function is to maximize: $c'_e = -c_e$
2. We can relax the assumption that the c_e are all distinct, by adding extremely small perturbations so that you can distinguish between edges that the same and the property of being optimal or not is preserved.

3 Project Ideas

1. These algorithms have a nice interpretation and analysis in terms of linear programming duality and primal-dual pairs.
2. Our study of trees is a special case of the study of certain objects called "matroids," which generalize linear independence (from linear algebra).
    - A matroid is a set system $M = (X, \mathcal{I})$, where X is a finite set and $\mathcal{I}$ is a collection of subsets of X such that:
        (a) if $I \in \mathcal{I}$ and $J \subseteq I$, then $J \in \mathcal{I}$, and
        (b) if $I, J \in \mathcal{I}$ are such that $|I| < |J|$, then there exists $z \in J \setminus I$ such that $I + z \in \mathcal{I}$.

In the language of trees and spanning trees:

- X is the set of edges
- $\mathcal{I}$ is the set of acyclic subsets of edges

The second property is equivalent to the "exchange" argument we used today. It can be shown that greedy algorithms are optimal if and only if you are optimizing over a matroid. Matroids are a more advanced topic in combinatorics; you are likely to encounter them again in a more advanced course in combinatorial optimization.