CS 6505: Computability & Algorithms
Lectures 3 & 4, January 20 & 22, 2010

Greedy Algorithms

In this lecture we will examine a couple of famous greedy algorithms and then look at matroids, which are a class of structures that can be solved by greedy algorithms.

Examples of Greedy Algorithms

What are some examples of greedy algorithms?

Maximum Matching: A **matching** is a set of edges in a graph that do not share vertices. Another way of thinking about this is that vertices are paired up so that we don't use the same vertex twice. A **maximal matching** is a pairing that involves as many vertex pairs as possible.

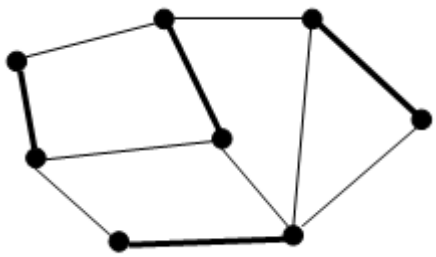

Figure 1: Example Matching: Matching consists of edges in bold.

Maximum Weight Independent Set: Suppose we are given a set of vectors in $\mathbf{R}^n$ and a weight for each such vector. Then find a linearly independent subset of these vectors of maximum total weight. (This was part of an exercise in Friday's lecture on January 22nd.)

A natural algorithm in all these cases is to incrementally build a solution; at each step, choose a new element that provides the most benefit. However, we cannot blindly choose new elements without some thought. For example, when choosing edges for a maximum matching, just picking edges greedily may not give a maximum cardinality matching (i.e., a matching with the most edges). When solving the Maximum Weight Independent Set problem, we could keep on picking the "heaviest" vectors that maintain linear independence. We saw on the following lecture whether this worked or not.

Kruskal's Algorithm

Let's take a look at a classic graph problem. A graph $G = (V, E)$ is a collection of vertices (or "nodes") V with a set of edges E. Now suppose that we have a cost function $c : E \to \mathbb{R}_+$ that maps each edge to a positive real number. We now want to find the minimum-cost subgraph (or subset of edges) that contains a path between any two vertices.

Before we find such a solution, what is the structure of such a subgraph? Such a subgraph has no cycles. Removing an edge of a cycle maintains at least one path between any two vertices. Such an acyclic subgraph that visits all vertices is a spanning tree.

How many edges will such a subgraph have?

Claim: If a connected graph has n vertices, then any spanning tree has $n - 1$ edges.

Claim: Any connected graph with n vertices and $n - 1$ edges is a tree.

Now we will show how to find the minimum spanning tree.

(Kruskal) Greedy Algorithm to Build a Minimum Spanning Tree (MST): First, sort edges in increasing order of cost. Continue to pick edges in this order without creating a cycle. When all vertices have been selected, then a minimum spanning tree has been created.

Now we must determine if this will work. Before we can do that, we need to define a cut. A **cut** of a graph $G = (V, E)$ is a partition of G's vertices into at least two sets.

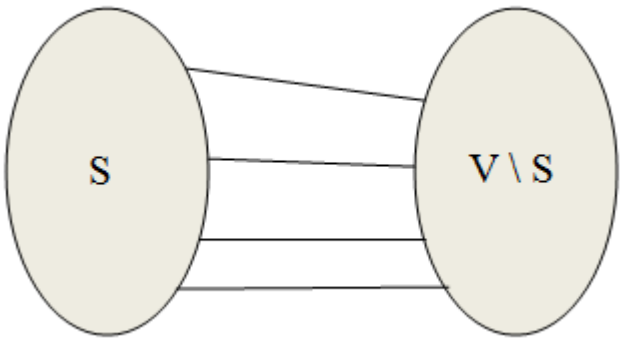

Figure 2: Example Cut: The left side of the graph is the set S, and the right side of the graph is all of the vertices except S (or $V \backslash S$). The cut consists of the 4 edges between the two components of the graph.

Lemma: For any cut, consider any minimum weight edge e in the cut. Then there is some minimum spanning tree that contains e. (Note that our wording considers the possibility that there are many edges in the cut that have the same minimum weight.)

Proof Take an MST T on G. Suppose that there is a cut in G such that e has minimum weight among edges in the cut but $e \notin T$. Now consider the cycle induced by $T \cup \{e\}$. (Exercise: Why does adding an edge to a spanning tree induce a cycle?) Now remove any edge f of this cycle that is in the same cut. The edge e has weight less than or equal to the weight of f. Then $T' = T \cup \{e\} \backslash \{f\}$ has weight less than or equal to T, so T' is also an MST.

Theorem: Kruskal's algorithm gives an MST.

Proof: As each new edge is added, the previous lemma gives the property that at least one MST contains all the edges added so far. But why is the next edge the smallest in some cut? Suppose the next edge connects vertices u and v. Then some cut creates a component (i.e., a connected subgraph) that contains u and another component that contains v. This edge must have the lowest weight among edges that connect these two components across the cut.

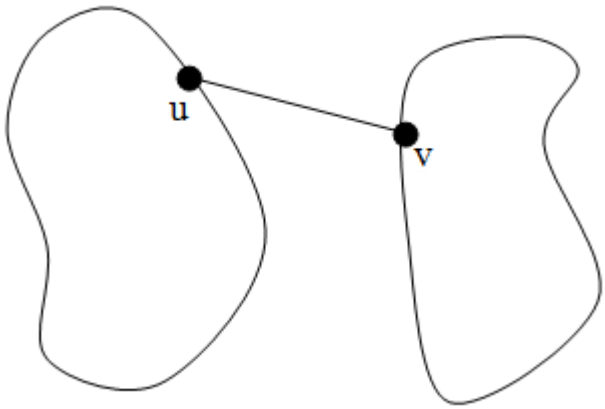

Figure 3: The lowest-cost edge happens to be between u and v. Then this edge is the lowest weight within some cut, as shown.

Prim's Algorithm: Starting at a node x, add a minimum edge incident to x (i.e., an edge that includes x as one of its endpoints). Repeat adding a minimum edge incident to the current tree which connects to a new vertex.

Theorem: Prim's algorithm yields an MST. The proof is based on the cut property discussed earlier.

Matroids

So when do greedy algorithms work? First, let's describe an abstract structure called a **matroid**.

Matroid: A matroid consists of a base set U and a collection I of independent subsets. Independence will be related to different objects depending on the problem - for the minimum spanning tree, an independent subset could be a tree. In linear algebra, an independent subset could be linearly independent vectors. A matroid's notion of independence is as follows. $A, B \subseteq U$ satisfy:

1. If A is a subset of B and B is independent subset, then A is independent subset, too. In other words:

$$A \subseteq B, B \in I \Rightarrow A \in I$$

2. The empty set is independent.

3. Suppose A and B are indepedent sets and A is smaller than B. Then there is some element e in B that is not in A such that A plus e is also an independent set. In other words:

$$A, B \in I, |A| < |B| \Rightarrow \exists e \in B \backslash A \text{ such that } A \cup \{e\} \in I$$

Now suppose that we are given any nonnegative weighting of elements of U. Another way of writing this is $w : U \rightarrow \mathbb{R}_+$. Finding a max weight independent set can be done using the greedy algorithm.

Proof Suppose the greedy algorithm gives us a solution

$$X = \{x_1, x_2, \ldots, x_n\}, x_1 \geq x_2 \geq \ldots \geq x_n$$

,

but the truly optimal solution is:

$$Y = \{y_1, y_2, \ldots, y_n\}, y_1 \geq y_2 \geq \ldots \geq y_n$$

.

Now suppose that k is the earliest index such that $x_k < y_k$.

Then $|\{x_1, \ldots x_{k-1}\}| < |\{y_1, \ldots y_k\}|$, and both sets are independent. By the second property of matroids, there is some y_j that can be added to $\{x_1, ..., x_{k-1}\}$ maintaining independence. We know that $w(y_j) > w(y_k)$ because $j < k$, and $w(y_k) > w(x_k)$ by assumption. This means that y_j would have been chosen by the greedy algorithm earlier than x_k. This is a contradiction, so the greedy algorithm must find the optimal solution.

(As an exercise, you may want to show that all maximum weight independent sets have the same cardinality (i.e., number of elements). You can use a small modification of the previous argument to show that this is true.)

Finally, not every greedy algorithm is associated with a matroid, but matroids do give an easy way to construct greedy algorithms for many problems.

Relevant Readings

- Kleinberg and Tardos, *Algorithm Design*, Chapter 4 (Greedy Algorithms). This book has an excellent treatment of greedy algorithms. It is highly recommended and is on reserve at the library.

- http://compgeom.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/04d-matroids.pdf
  (The tilde (~) may have to be retyped if you cut and paste this link.)

- Douglas West, *Introduction to Graph Theory, Second Edition*, Section 2.3 (Minimum Spanning Trees), Section 8.2 (Matroids).

  The section on Matroids is more geared towards graph theory, but it may interest those who want a deeper understanding of them. *See Scott for a copy of this book; there is also one copy in the library.*