The problem is that we attempt to solve the simplest questions cleverly,
thereby rendering them unusually complex.
One should seek the simple solution.

— Anton Pavlovich Chekhov (c. 1890)

I love deadlines. I like the whooshing sound they make as they fly by.

— Douglas Adams, *The Salmon of Doubt* (2002)

E

Matroids

[Read Chapter 4 and 7 first.]

Status: Needs significant revision or deletion

Merge into greedy notes as hardsection?

E.1 Definitions

Many problems that can be correctly solved by greedy algorithms can be described in terms of an abstract combinatorial object called a *matroid*. Matroids were first described in 1935 by the mathematician Hassler Whitney as a combinatorial generalization of linear independence of vectors—"matroid" means "something sort of like a matrix".

A matroid $(S, \mathcal{I})$ consists of a finite ***ground set*** S and a collection $\mathcal{I}$ of subsets of X that satisfies three axioms:

- **Non-emptiness:** The empty set is in $\mathcal{I}$. (Thus, $\mathcal{I}$ is not itself empty.)
- **Heredity:** If $\mathcal{I}$ contains a subset $X \subseteq S$, then $\mathcal{I}$ contains every subset of X.
- **Exchange:** If X and Y are two sets in $\mathcal{I}$ where $|X| > |Y|$, then $Y \cup \{x\} \in \mathcal{I}$ for some element $x \in X \setminus Y$.

The subsets in $\mathcal{I}$ are typically called ***independent sets***; for example, we would say that any subset of an independent set is independent. An independent set is called a ***basis*** of the matroid if it is not a proper subset of another independent set. The exchange property implies that every basis of a matroid has the same cardinality. The ***rank*** of a subset X of the ground set is the size of the largest independent subset of X. A subset of S that is not in $\mathcal{I}$ is called ***dependent*** (surprise, surprise). Finally, a dependent set is called a ***circuit*** if every proper subset is independent.

Most of this terminology is justified by Whitney's original example:

- ***Linear matroid:*** Let A be any $n \times m$ matrix. A subset $I \subseteq \{1, 2, \ldots, n\}$ is independent if and only if the corresponding subset of columns of A is linearly independent.

The heredity property follows directly from the definition of linear independence; the exchange property is implied by an easy dimensionality argument. A basis in any linear matroid is also a basis (in the linear-algebra sense) of the vector space spanned by the columns of A. Similarly, the rank of a set of indices is precisely the rank (in the linear-algebra sense) of the corresponding set of column vectors.

Here are several other examples of matroids; some of these we will see again later. I will leave the proofs that these are actually matroids as exercises for the reader.

- ***Uniform matroid $U_{k,n}$:*** A subset $X \subseteq \{1, 2, \ldots, n\}$ is independent if and only if $|X| \leq k$. Any subset of $\{1, 2, \ldots, n\}$ of size k is a basis; any subset of size $k+1$ is a circuit.
- ***Graphic/cycle matroid $\mathcal{M}(G)$:*** Let $G = (V, E)$ be an arbitrary undirected graph. A subset of E is independent if it defines an *acyclic* subgraph of G. A basis in the graphic matroid is a spanning tree of G; a circuit in this matroid is a cycle in G.
- ***Cographic/cocycle matroid $\mathcal{M}^*(G)$:*** Let $G = (V, E)$ be an arbitrary undirected graph. A subset $I \subseteq E$ is independent if the complementary subgraph $(V, E \setminus I)$ of G is *connected*. A basis in this matroid is the complement of a spanning tree; a circuit in this matroid is a *cocycle*—a minimal set of edges that disconnects the graph.
- ***Matching matroid:*** Let $G = (V, E)$ be an arbitrary undirected graph. A subset $I \subseteq V$ is independent if there is a matching in G that covers I.
- ***Disjoint path matroid:*** Let $G = (V, E)$ be an arbitrary *directed* graph, and let s be a fixed vertex of G. A subset $I \subseteq V$ is independent if and only if there are edge-disjoint paths from s to each vertex in I.

Now suppose each element of the ground set of a matroid $\mathcal{M}$ is given an arbitrary non-negative weight. The ***matroid optimization problem*** is to compute a basis with maximum total weight. For example, if $\mathcal{M}$ is the cycle matroid for a graph G, the matroid optimization problem asks us to find the maximum spanning tree of G. Similarly, if $\mathcal{M}$ is the cocycle matroid for G, the matroid optimization problem seeks (the complement of) the *minimum* spanning tree.

The following natural greedy strategy computes a basis for any weighted matroid $(S, \mathcal{I})$, where the ground set S is represented by an array $S[1..n]$, and the weights of ground elements are represented by another array $w[1..n]$.

```
GreedyBasis(S[1..n], I, w[1..n]):
  sort S in decreasing order of weight w
  G ← ∅
  for i ← 1 to n
      if G ∪ {S[i]} ∈ I
          add S[i] to G
  return G
```

Suppose we can test in $F(n)$ time whether a given subset $X \subset S$ is independent. Then this algorithm runs in $O(n \log n + n \cdot F(n))$ time.

Theorem E.1. *For any matroid $\mathcal{M} = (S, \mathcal{I})$ and any weight function w, GreedyBasis$(S, \mathcal{I}, w)$ returns a maximum-weight basis of $\mathcal{M}$.*

Proof: We use a standard exchange argument. Let $G = \{g_1, g_2, \ldots, g_k\}$ be the independent set returned by GreedyBasis$(S, \mathcal{I}, w)$. If any other element of S could be added to G to obtain a larger independent set, the greedy algorithm would have added it. Thus, G is a basis.

For purposes of deriving a contradiction, suppose there is an independent set $H = \{h_1, h_2, \ldots, h_\ell\}$ such that

$$\sum_{i=1}^{k} w(g_i) < \sum_{j=1}^{\ell} w(h_i).$$

Without loss of generality, assume that H is a basis. The exchange property now implies that $k = \ell$.

Now suppose the elements of G and H are indexed in order of decreasing weight. Let i be the smallest index such that $w(g_i) < w(h_i)$, and consider the independent sets

$$G_{i-1} = \{g_1, g_2, \ldots, g_{i-1}\} \quad \text{and} \quad H_i = \{h_1, h_2, \ldots, h_{i-1}, h_i\}.$$

By the exchange property, there is some element $h_j \in H_i$ such that $G_{i-1} \cup \{h_j\}$ is an independent set. We have $w(h_j) \geq w(h_i) > w(g_i)$. Thus, the greedy algorithm considers *and rejects* the heavier element h_j before it considers the lighter element g_i. But this is impossible—the greedy algorithm accepts elements in decreasing order of weight. □

We now immediately have a correct greedy optimization algorithm for *any* matroid. Returning to our examples:

- Linear matroid: Given a matrix A, compute a subset of vectors of maximum total weight that span the column space of A.

- Uniform matroid: Given a set of weighted objects, compute its k largest elements.
- Cycle matroid: Given a graph with weighted edges, compute its maximum spanning tree. In this setting, the greedy algorithm is better known as *Kruskal's algorithm*.
- Cocycle matroid: Given a graph with weighted edges, compute its minimum spanning tree.
- Matching matroid: Given a graph, determine whether it has a perfect matching.
- Disjoint path matroid: Given a directed graph with a special vertex s, find the largest set of edge-disjoint paths from s to other vertices.

The exchange condition for matroids turns out to be crucial for the success of this algorithm. A *subset system* is a finite collection $\mathcal{S}$ of finite sets that satisfies the heredity condition—If $X \in \mathcal{S}$ and $Y \subseteq X$, then $Y \in \mathcal{S}$—but not necessarily the exchange condition.

Theorem E.2. *For any subset system $\mathcal{S}$ that is **not** a matroid, there is a weight function w such that* GREEDYBASIS$(\mathcal{S}, w)$ *does **not** return a maximum-weight set in $\mathcal{S}$.*

Proof: Let X and Y be two sets in $\mathcal{S}$ that violate the exchange property—$|X| > |Y|$, but for any element $x \in X \setminus Y$, the set $Y \cup \{x\}$ is not in $\mathcal{S}$. Let $m = |Y|$. We define a weight function as follows:

- Every element of Y has weight $m + 2$.
- Every element of $X \setminus Y$ has weight $m + 1$.
- Every other element of the ground set has weight zero.

With these weights, the greedy algorithm will consider and accept every element of Y, then consider and reject every element of X, and finally consider all the other elements. The algorithm returns a set with total weight $m(m+2) = m^2 + 2m$. But the total weight of X is at least $(m+1)^2 = m^2 + 2m + 1$. Thus, the output of the greedy algorithm is not the maximum-weight set in $\mathcal{S}$. □

Recall the Applied Chaos scheduling problem considered in Chapter 4. There is a natural subset system associated with this problem: A set of classes is independent if and only if no two classes overlap. (This is just the graph-theory notion of "independent set"!) This subset system is *not* a matroid, because there can be maximal independent sets of different sizes, which violates the exchange property. If we consider a *weighted* version of the class scheduling problem, where each class is worth a different number of hours, Theorem **??** implies that the greedy algorithm will *not* always find the optimal schedule. (In fact, there's an easy counterexample with only two classes!) However, Theorem **??** does *not* contradict the correctness of the greedy algorithm for the original *unweighted* problem; that problem uses a particularly lucky choice of weights (all equal).

E.2 Scheduling with Deadlines

Suppose you have n tasks to complete in n days; each task requires your attention for a full day. Each task comes with a *deadline*, the last day by which the job should be completed, and a *penalty* that you must pay if you do not complete each task by its assigned deadline. What order should you perform your tasks in to minimize the total penalty you must pay?

More formally, you are given an array $D[1..n]$ of deadlines an array $P[1..n]$ of penalties. Each deadline $D[i]$ is an integer between 1 and n, and each penalty $P[i]$ is a non-negative real number. A *schedule* is a permutation of the integers $\{1, 2, \ldots, n\}$. The scheduling problem asks you to find a schedule π that minimizes the following cost:

$$cost(\pi) := \sum_{i=1}^{n} P[i] \cdot [\pi(i) > D[i]].$$

This doesn't look anything like a matroid optimization problem. For one thing, matroid optimization problems ask us to find an optimal *set*; this problem asks us to find an optimal *permutation*. Surprisingly, however, this scheduling problem is actually a matroid optimization in disguise! For any schedule π, call tasks i such that $\pi(i) > D[i]$ *late*, and all other tasks *on time*. The following trivial observation reveals the underlying matroid structure.

> The cost of a schedule is determined by the subset of tasks that are on time.

Call a subset X of the tasks ***realistic*** if there is a schedule π in which every task in X is on time. We can precisely characterize the realistic subsets as follows. Let $X(t)$ denote the subset of tasks in X whose deadline is on or before t:

$$X(t) := \{i \in X \mid D[i] \le t\}.$$

In particular, $X(0) = \emptyset$ and $X(n) = X$.

Lemma E.3. *Let $X \subseteq \{1, 2, \ldots, n\}$ be an arbitrary subset of the n tasks. X is realistic if and only if $|X(t)| \le t$ for every integer t.*

Proof: Let π be a schedule in which every task in X is on time. Let i_t be the tth task in X to be completed. On the one hand, we have $\pi(i_t) \ge t$, since otherwise, we could not have completed $t-1$ other jobs in X before i_t. On the other hand, $\pi(i_t) \le D[i]$, because i_t is on time. We conclude that $D[i_t] \ge t$, which immediately implies that $|X(t)| \le t$.

Now suppose $|X(t)| \le t$ for every integer t. If we perform the tasks in X in increasing order of deadline, then we complete all tasks in X with deadlines t or less by day t. In particular, for any $i \in X$, we perform task i on or before its deadline $D[i]$. Thus, X is realistic. □

We can now define a ***canonical schedule*** for any set X as follows: execute the tasks in X in increasing deadline order, and then execute the remaining tasks in any order. The previous proof implies that a set X is realistic if and only if every task in X is on time in the canonical schedule for X. Thus, our scheduling problem can be rephrased as follows:

> Find a realistic subset X such that $\sum_{i \in X} P[i]$ is maximized.

So we're looking for optimal subsets after all!

Lemma E.4. *The collection of realistic sets of jobs forms a matroid.*

Proof: The empty set is vacuously realistic, and any subset of a realistic set is clearly realistic. Thus, to prove the lemma, it suffices to show that the exchange property holds. Let X and Y be realistic sets of jobs with $|X| > |Y|$.

Let t^* be the largest integer such that $|X(t^*)| \le |Y(t^*)|$. This integer must exist, because $|X(0)| = 0 \le 0 = |Y(0)|$ and $|X(n)| = |X| > |Y| = |Y(n)|$. By definition of t^*, there are more tasks with deadline $t^* + 1$ in X than in Y. Thus, we can choose a task j in $X \setminus Y$ with deadline $t^* + 1$; let $Z = Y \cup \{j\}$.

Let t be an arbitrary integer. If $t \le t^*$, then $|Z(t)| = |Y(t)| \le t$, because Y is realistic. On the other hand, if $t > t^*$, then $|Z(t)| = |Y(t)| + 1 \le |X(t)| < t$ by definition of t^* and because X is realistic. The previous lemma now implies that Z is realistic. This completes the proof of the exchange property. □

This lemma implies that our scheduling problem is actually a matroid optimization problem, so the greedy algorithm finds the optimal schedule.

```
GREEDYSCHEDULE(D[1..n], P[1..n]):
  Sort P in increasing order, and permute D to match
  j ← 0
  for i ← 1 to n
      X[j + 1] ← i
      if X[1..j + 1] is realistic
          j ← j + 1
  return the canonical schedule for X[1..j]
```

To turn this outline into a real algorithm, we need a procedure to test whether a given subset of jobs is realistic. Lemma 9 immediately suggests the following strategy to answer this question in $O(n)$ time.

```
REALISTIC?(X[1..m], D[1..n]):
  ⟨⟨X is sorted by increasing deadline: i ≤ j ⟹ D[X[i]] ≤ D[X[j]]⟩⟩
  N ← 0
  j ← 0
  for t ← 1 to n
      if D[X[j]] = t
          N ← N + 1
          j ← j + 1
      ⟨⟨Now N = |X(t)|⟩⟩
      if N > t
          return FALSE
  return TRUE
```

If we use this subroutine, GREEDYSCHEDULE runs in $O(n^2)$ time. By using some appropriate data structures, the running time can be reduced to $O(n \log n)$; details are left as an exercise for the reader.

Exercises

1. Prove that for any graph G, the "graphic matroid" $\mathcal{M}(G)$ is in fact a matroid. (This problem is really asking you to prove that Kruskal's algorithm is correct!)

2. Prove that for any graph G, the "cographic matroid" $\mathcal{M}^*(G)$ is in fact a matroid.

♦3. Prove that for any graph G, the "matching matroid" of G is in fact a matroid. *[Hint: What is the symmetric difference of two matchings?]*

♦4. Prove that for any directed graph G and any vertex s of G, the resulting "disjoint path matroid" of G is in fact a matroid. *[Hint: This question is **much** easier if you're already familiar with maximum flows.]*

5. Let G be an undirected graph. A set of cycles $\{c_1, c_2, \ldots, c_k\}$ in G is called *redundant* if every edge in G appears in an even number of c_i's. A set of cycles is *independent* if it contains no redundant subset. A maximal independent set of cycles is called a *cycle basis* for G.

   (a) Let C be any cycle basis for G. Prove that for any cycle γ in G, there is a subset $A \subseteq C$ such that $A \cap \{\gamma\}$ is redundant. In other words, γ is the 'exclusive or' of the cycles in A.

   (b) Prove that the set of independent cycle sets form a matroid.

   ♥(c) Now suppose each edge of G has a weight. Define the weight of a cycle to be the total weight of its edges, and the weight of a *set* of cycles to be the total weight of all cycles in the set. (Thus, each edge is counted once for every cycle in

which it appears.) Describe and analyze an efficient algorithm to compute the minimum-weight cycle basis in G.

6. Describe a modification of GREEDYSCHEDULE that runs in $O(n \log n)$ time. *[Hint: Store X in an appropriate data structure that supports the operations "Is $X \cup \{i\}$ realistic?" and "Add i to X" in $O(\log n)$ time each.]*