CS5275 Lecture 5: Submodularity

Jonathan Scarlett

December 6, 2024

Useful references:

- Krause and Golovin, "Submodular Function Maximization"
- Columbia lecture notes: `https://www.columbia.edu/~yf2414/ln-submodular.pdf`
- Various research papers linked throughout the sections below

1 Introduction

In discrete optimization, it is common to encounter problems that consist of *choosing elements from a set*:

- Choosing items in resource allocation and similar problems (e.g., knapsack);
- Choosing subsets of nodes in graph problems (e.g., minimum cut);
- Choosing representative data points in data summarization;
- etc.

In this lecture, we interpret such problems as optimizing some $f(S)$ over $S \subseteq \{1, \dots, n\}$. Although we will not use it much, this could also equivalently be viewed as optimizing $f(\mathbf{x})$ over $\mathbf{x} \in \{0,1\}^n$; the equivalence is seen by simply interpreting S as $\{i : x_i = 1\}$.

Set function maximization: Find "best" subset S of ground set $V = \{1, \dots, n\}$.

- Unconstrained:
$$\text{maximize}_{S \subseteq V} \ f(S)$$
- Constrained:
$$\text{maximize}_{S \in \mathcal{S}} \ f(S)$$
where the constraint set $\mathcal{S}$ contains subsets of V. We will especially focus on the constraint set $\mathcal{S} = \{S : |S| \le k\}$ that enforces choosing at most k elements for some $k > 0$.

Much like continuous functions, assumptions are needed on f to give us hope of being able to perform optimization efficiently. We will consider the following two main assumptions, with an emphasis on the second one (submodularity).

Definition. A set function f is said to be *monotone* if, for all $S \subseteq T \subseteq V$, it holds that $f(S) \le f(T)$.

- Note: When we need to specifically emphasize the direction of monotonicity, we will use the terminology *non-decreasing* or *non-increasing*. But unless explicitly specified otherwise, the term *monotone* will mean non-decreasing in this lecture.

Definition. A real-valued set function f is said to be *submodular* if, for all $S \subseteq T \subseteq V$ and $e \in V \backslash T$, it holds that

$$\Delta(e|S) \geq \Delta(e|T),$$

where we define $\Delta(e|S) = f(S \cup \{e\}) - f(S)$ (i.e., the gain of adding e to S), and similarly for $\Delta(e|T)$.

- Intuition: "Diminishing returns" – adding to a smaller set gains you more. This intuition is especially useful for monotone functions (it can get a bit less easy to picture in the general case).
- Related notions: (i) The function is *modular* if it always holds that

  $$\Delta(e|S) = \Delta(e|T)$$

  It is not hard to see that this is equivalent to "linear", say $f(S) = \sum_{i \in S} c_i$, which is equivalent to $c^T x$ with $x \in \{0,1\}^n$ and $S = \{i : x_i = 1\}$.

  (ii) The function is *supermodular* if it always holds that

  $$\Delta(e|S) \leq \Delta(e|T).$$

  We will focus on submodularity in this lecture. The types of problems we will solve below will be fairly trivial for modular (linear) functions, e.g., solved by sorting n numbers and taking the k highest ones.

Equivalent definitions of submodularity. For any function $f : 2^V \to \mathbb{R}$, all of the following are equivalent:

(i) $\Delta(e|S) \geq \Delta(e|T)$ for all $S \subseteq T$ and all $e \in V \setminus T$ (this is a repeat of the above definition)

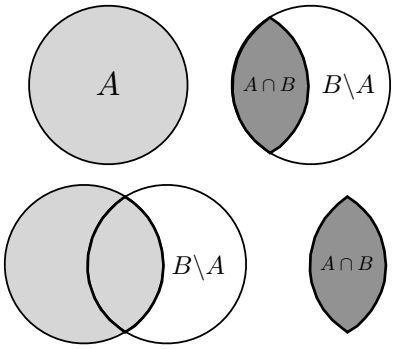
(ii) $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$ for all S, T

(iii) $\Delta(e|S) \geq \Delta(e|S \cup \{e'\})$ for all sets S and elements e, e'

(iv) (In the case that f is monotone) $f(T) \leq f(S) + \sum_{e \in T \setminus S} \Delta(e|S)$ for all S, T

We will not prove all of these, but we proceed to give some intuition and some details as follows:

- Definition (iii) is just (i) applied to $T = S \cup \{e'\}$, and it is not difficult to show that the property for $|T| = |S| + 1$ implies the general case. Definition (iv) just states that the overall gain of adding $T \setminus S$ is no larger than the individual gains of its elements, which is natural given diminishing returns.
- Definition (ii) may seem less intuitive, but can be understood visually as comparing what happens when we add $T \setminus S$ to a smaller set ($S \cap T$) vs. a larger set (S) – see the figure below. By the diminishing returns property, the former should cause a higher increase in f.



- There are fairly elementary proofs of going from one definition to any other equivalent one. For example, the following analysis shows that (iii) implies (ii): Fix any S, T, let $W = S \cap T$, and denote $S\backslash T = \{j_1, \ldots, j_s\}$ and $T\backslash S = \{k_1, \ldots, k_t\}$. Then

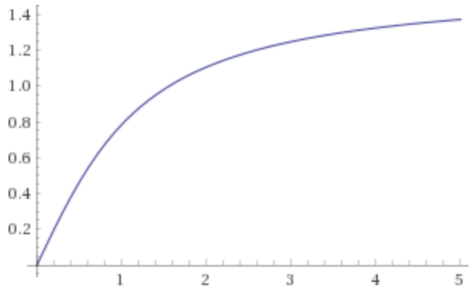
$$\begin{aligned}
& f(T) - f(S \cap T) \\
= \; & f(W \cup \{k_1, \ldots, k_t\}) - f(W) \qquad \text{(def. } W \text{ and } k_i\text{)} \\
= \; & \sum_{i=1}^{t} \Big(f(W \cup \{k_1, \ldots, k_i\}) - f(W \cup \{k_1, \ldots, k_{i-1}\}) \Big) \qquad \text{(telescoping)} \\
\geq \; & \sum_{i=1}^{t} \Big(f(W \cup \{k_1, \ldots, k_i, j_1\}) - f(W \cup \{k_1, \ldots, k_{i-1}, j_1\}) \Big) \qquad \text{(using (iii))} \\
\vdots \; & \\
\geq \; & \sum_{i=1}^{t} \Big(f(W \cup \{k_1, \ldots, k_i, j_1, \ldots, j_s\}) - f(W \cup \{k_1, \ldots, k_{i-1}, j_1, \ldots, j_s\}) \Big) \qquad \text{using (iii)} \\
= \; & \sum_{i=1}^{t} \Big(f(S \cup \{k_1, \ldots, k_i\}) - f(S \cup \{k_1, \ldots, k_{i-1}\}) \Big) \qquad \text{(definition of } W \text{ and } j_{(\cdot)}\text{)} \\
= \; & f(S \cup T) - f(S) \qquad \text{(telescoping).}
\end{aligned}$$

- As one more example, here is a proof that (i) implies (iv) when f is monotone: Again letting $T\backslash S = \{k_1, \ldots, k_t\}$, we have

$$\begin{aligned}
f(T) \; & \leq \; f(S \cup T) \qquad \text{(monotonicity)} \\
& = \; f(S) + (f(S \cup T) - f(S)) \\
& = \; f(S) + \sum_{i=1}^{t} \Big(f(S \cup \{k_1, \ldots, k_i\}) - f(S \cup \{k_1, \ldots, k_{i-1}\}) \Big) \qquad \text{(telescoping)} \\
& \leq \; f(S) + \sum_{i=1}^{t} (f(S \cup \{k_i\}) - f(S)). \qquad \text{(using (i))}
\end{aligned}$$

Relations to convexity and concavity: Submodular functions share some properties of *concave functions*:

- Diminishing returns (see the figure below)
- (Non-monotone case) Any local maximum is within a factor of $\frac{1}{2}$ of globally optimal



- Maximization (constrained or unconstrained) can be done *approximately* using computationally efficient algorithms
- A function of the form $f(S) = g(|S|)$ is submodular if g is concave (see below for a more general statement).

...but they also share some properties with *convex functions*:

- Unconstrained minimization can be done *exactly* using computationally efficient algorithms (constrained not so!)
- A natural extension from sets (represented by $\{0,1\}$-values) to continuous values (i.e., $[0,1]$) called the *Lovász extension* is a convex function

Sometimes neither concave-like not convex-like properties apply (e.g., the pointwise max. or min. of two submodular functions may not be submodular). A classical paper on the above connections is "Submodular functions and convexity" (Lovász, 1983).

Properties of Submodular Functions: Let f_1 and f_2 be submodular functions. Then:

1. **Linear combinations:** If c_1, c_2 are positive, then $f(S) = c_1 f_1(S) + c_2 f_2(S)$ is submodular.
2. **Concave of modular:** If $g : 2^V \to \mathbb{R}$ is modular and $h : \mathbb{R} \to \mathbb{R}$ is concave, then $f(S) = h(g(S))$ is submodular. An important special case is $g(S) = |S|$.
   - Intuition: At least for monotone g, this can be understood from diminishing returns
3. **Residual:** $f(S) = f_1(S \cup B) - f_1(B)$ is submodular for any B.
   - Intuition: This function measures how much S increases f_1 after the elements of B have already been chosen. The subtraction of $f_1(B)$ is optional, but ensures $f_1(\emptyset) = 0$.
4. **Conditioning:** $f(S) = f_1(S \cap A)$ is submodular for any A.
   - Intuition: Items from A get added as usual, but those outside A play no role.
5. **Reflection:** $f(S) = f_1(V \backslash S)$ is submodular.
   - Intuition: This is analogous to how if $g(x)$ is convex, so is $g(c - x)$.
6. **Truncation:** If f_1 is also monotone, then $f(S) = \min\{c, f_1(S)\}$ is submodular for any $c \in \mathbb{R}$.

7. **Minimum:** Although $f(S) = \min\{f_1(S), f_2(S)\}$ is not submodular in general, it is submodular when either $f_1 - f_2$ or $f_2 - f_1$ is monotone.

Some of these properties (e.g., linear combinations, residual, conditioning) can be proved essentially directly from the definition of submodularity. Here are a couple of proofs for the "less obvious" properties:

- Proof of reflection property: Fix any sets S and T, and let $W = V \setminus (S \cup T)$. Using the residual property (without the subtracted term, which we won't need), the function $g(A) = f_1(A \cup W)$ is submodular, so definition (ii) of submodularity gives

$$g(A) + g(B) \geq g(A \cup B) + g(A \cap B).$$

  Now let $f(S) = f_1(V \setminus S)$ and consider the following:

$$f(S) + f(T) = f_1(V \setminus S) + f_1(V \setminus T)$$
$$f(S \cap T) + f(S \cup T) = f_1(V \setminus (S \cap T)) + f_1(V \setminus (S \cup T))$$

  Letting $A = S \setminus T$ and $B = T \setminus S$, it is straightforward to check via the above definitions that $f_1(V \setminus S) = g(B)$, $f_1(V \setminus T) = g(A)$, $f_1(V \setminus (S \cap T)) = g(A \cup B)$, and $f_1(V \setminus (S \cup T)) = g(A \cap B)$ (in fact $A \cap B = \emptyset$). Therefore, by the above 3 displayed equations, submodularity of f (i.e., $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$) is a consequence of submodularity of g.

- Proof of truncation property: Let $\Delta_1(e|S) = f_1(S \cup \{e\}) - f_1(S)$ and $\Delta_f(e|S) = f(S \cup \{e\}) - f(S)$, where $f(S) = \min\{c, f_1(S)\}$. We use the first definition of submodularity, stating that $\Delta_1(e|S) \geq \Delta_1(e|T)$ when $S \subseteq T$.

  We wish to show that $\Delta_f(e|S) \geq \Delta_f(e|T)$ as well. Notice that if $f(T) = c$ then adding further items will just keep the value at c (by monotonicity of f_1), giving $\Delta_f(e|T) = 0$ and making the desired inequality trivial. So we can assume $f(T) < c$, which implies $f(S) < c$ by monotonicity. Then we have

$$\Delta_f(e|S) = f(S \cup \{e\}) - f(S) = \min\{c, f_1(S \cup \{e\})\} - f_1(S) = \min\{c - f_1(S), \Delta_e(e|S)\},$$

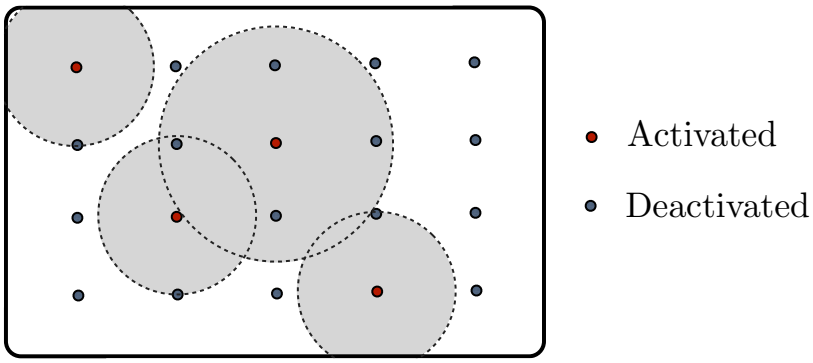
  and similarly for $\Delta_f(e|T)$. The desired inequality follows since $c - f_1(S) \geq c - f_1(T)$ (by monotonicity) and $\Delta_e(e|S) \geq \Delta_e(e|T)$ (by submodularity).

2 Examples of Submodular Functions

Example 1: Coverage functions

$$f(S) = \text{Area covered by activating all sensors in } S$$

- This is clearly monotone (activating more sensors implies more coverage) and submodular (the more sensors were activated already, the less gain there will be by activating another one)
- If we replace the continuous notion of "area" by the discrete notion of "cardinality", we get the *set cover* problem. A special case of set cover is *vertex cover*, which is the (NP-hard) problem of finding a set of vertices that includes at least one endpoint of every edge. This is related to maximizing $f(S)$, defined



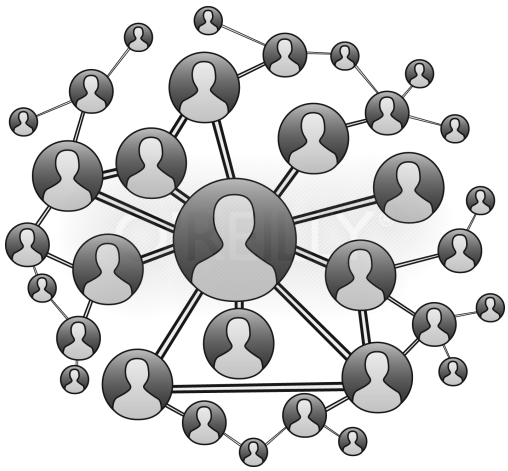
to be the number of edges connected to at least one node in S. (Specifically, we are interested in how small S can be to get $f(S) = |E|$, the total number of edges.)

Example 2: Let $\mathbf{X}$ be a matrix, let V be the indices of its columns, and let $\mathbf{X}_S$ be the submatrix formed by keeping only the columns indexed by S. Then $r(S) = \text{rank}(\mathbf{X}_S)$ is *monotone submodular*.

- Recall that rank can be interpreted as the maximum number of linearly independent columns (or rows).
- The intuition on monotonicity: Adding another column can either keep that maximum the same or increase it by one.
- The intuition on submodularity: Adding a column to a larger set means that there are more ways in which the column could already be a linear combination of the existing ones.

Example 3: Influence maximization

$$f(S) = \text{Total number of users influenced by advertising to } S$$

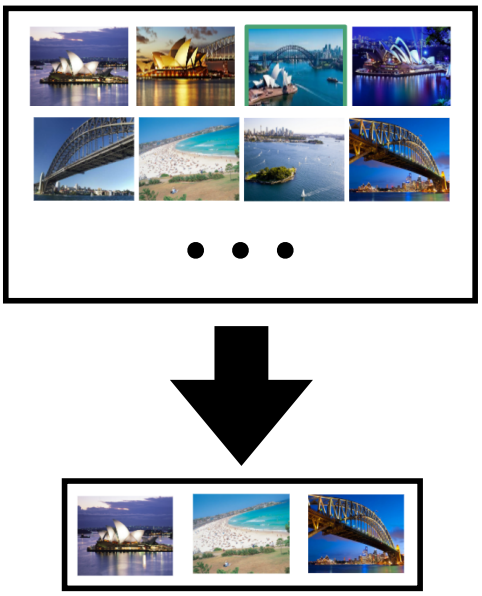


- There are several ways that influence could reasonably be measured (many of which lead to submodularity, but not all).

- Perhaps the simplest model is that given the graph $G = (V, E)$, advertising to some $v \in V$ amounts to *influencing v and its direct neighbors*. This leads to $f(S)$ being submodular in the same way as Example 1.
- There are much more sophisticated models, such as v influencing each of its neighbors independently with probability p, and each of those influenced people subsequently influencing their own neighbors with some (possibly smaller) probability, etc., still leading to submodularity if $f(S)$ is taken to be the *average* number of people influenced.
- See (e.g.,) the paper "Maximizing the spread of influence through a social network" for more on this application if you are interested.

Example 4: Data summarization

$$f(S) = \text{Representativeness of images in } S$$



- Again, there are many reasonable measures of "representativeness", some of which are submodular and some of which are not.
- For example, one way of measuring representativeness (and indeed giving submodularity) is to try to make every image in the full set V close to the nearest image in S:

$$g(S) = \frac{1}{n} \sum_{v \in V} \min_{s \in S} d(s, v),$$

  where $d(s, v)$ is a pre-specified distance measure between two images s and v. The minimizing s in the summation above is sometimes referred to as a *medoid*.
- The more representative S is, the smaller g is, so we could pose the data summarization problem as maximizing $f(S) = -g(S)$. To make this well-defined even when $S = \emptyset$, it's more common to further

define a *reference element* e_0 and instead consider

$$f(S) = g(\{e_0\}) - g(S \cup \{e_0\}),$$

which is monotone, submodular, and satisfies $f(\emptyset) = 0$ – see the appendix for a related exercise from which you can prove this.

- See (e.g.,) the paper "A class of submodular functions for document summarization" for more on this application if you are interested.

Example 5: Graph cut

- Given a graph $G = (V, E)$, split the nodes V into (S, S^c)
- Graph cut function $f(S)$: Number of edges between S and S^c
- This is *submodular* but *non-monotone* – see the appendix for the proof.
- Maximizing this $f(S)$ is the Maximum Cut (MAXCUT) problem, which is known to be NP-hard.

Example 6: The *entropy* of a random variable is defined as

$$H(X) = \sum_x P_X(x) \log \frac{1}{P_X(x)}. \tag{1}$$

Let $\mathbf{X} = (X_1, \ldots, X_n)$ be a random vector, and let $X_S = \{X_i\}_{i \in S}$. Then the entropy $f(S) = H(\mathbf{X}_S)$ is *monotone submodular*.

- Roughly speaking, $H(\mathbf{X}_S)$ measures the amount of uncertainty in the random variables indexed by S.
- Intuition on monotonicity: Adding more variables amounts to having higher uncertainty.
- Intuition on submodularity: Adding to a larger set increases the uncertainty less, because the existing variables provide information about the new variable (e.g., correlations or other dependencies).
- You may want to try proving monotonicity/submodularity after we have covered information theory.

3 Submodular Optimization

Several problems in theoretical computer science, game theory, machine learning, etc. can be cast as a submodular optimization problem.

Problem statement: Given a submodular function f and constraint set $\mathcal{S}$

$$\min_{S \in \mathcal{S}} f(S) \qquad \text{(SFMin)}$$

$$\max_{S \in \mathcal{S}} f(S) \qquad \text{(SFMax)}$$

If $\mathcal{S}$ contains all 2^n possible choices of S, the problem is said to be *unconstrained*.

For most of the lecture, we focus on the following:

- Maximization only
- Monotone functions only, typically "normalized" so that $f(\emptyset) = 0$
- Cardinality constraint: $\mathcal{S} = \{S : |S| \leq k\}$

Before doing so, we briefly mention some results regarding some of the other categories.

3.1 (**Optional**) Non-monotone submodular maximization

- Unconstrained SFMax admits a computationally efficient 1/2-*approximation* algorithm (i.e., an algorithm guaranteed to attain an f value within 1/2 of the highest possible). The factor 1/2 cannot be improved in general, in the sense that attaining a $(1/2 + \epsilon)$-approximation requires exponentially many function evaluations. See the research papers "A tight linear time (1/2)-approximation for unconstrained submodular maximization" and "Maximizing non-monotone submodular functions" for details.
- The cardinality-constrained (non-monotone) case has proved challenging; to my knowledge the best possible approximation constant is not known, but the paper "Submodular Maximization with Cardinality Constraints" shows (for non-negative f) that it is between $\frac{1}{e}$ and $\frac{1}{2}$, and equals $\frac{1}{2}$ in the limit of large n when the cardinality is $k = \frac{n}{2}$.
- Algorithms for this problem are also typically greedy-like but not quite as simple as the monotone setting (covered below), e.g., keeping track of two lists from which items are added/removed, using randomized methods for choosing additions and removals, or applying greedy steps to continuous extensions rather than directly on the discrete function.

3.2 (**Optional**) Submodular minimization

- Unconstrained submodular minimization turns out to be a problem that can be solved efficiently. A very brief outline is as follows:
  - The idea is to interpret $f(S)$ as $f(\mathbf{x})$, where $\mathbf{x} \in \{0,1\}^n$ (entry 1 if the element is in S, and 0 otherwise), then extend $f(\mathbf{x})$ to a function on $[0,1]^n$, minimize it "sufficiently accurately", and finally convert the solution back to a $\{0,1\}^n$-valued one.
  - The extension from $\{0,1\}^n$ to $[0,1]^n$ is done by defining $f(\mathbf{x}) = \mathbb{E}[f(S_\lambda(\mathbf{x}))]$ for $\lambda \sim \text{Uniform}[0,1]$, where $S_\lambda(\mathbf{x}) = \{i : x_i > \lambda\}$. This is called the *Lovász extension*.
  - Crucially, the Lovász extension is a convex function, which is why it is "easy" to minimize.
- In contrast, *constrained* submodular minimization is very hard, with special cases including NP-hard problems such as densest k-subgraph and uniform graph partitioning (which we won't describe here). In fact, in general it is hard to even obtain a solution that is optimal to within a constant factor (e.g., at most 1000 times the optimal value).

3.3 Cardinality-Constrained Submodular Maximization

- The cardinality-constrained problem is NP-hard in general (e.g., this can be shown via the special case of vertex cover). But it turns out that a simply greedy algorithm gives good approximation guarantees.

- **Greedy algorithm:**
  1. Initialize $S_0 = \emptyset$
  2. For $i = 1, \ldots, k$
     - Find $e_i = \arg\max_{e \in V \setminus S_{i-1}} \Delta(e|S_{i-1})$ (i.e, the element providing the largest gain)
     - Set $S_i = S_{i-1} \cup \{e_i\}$
  3. Return S_k

- **Theorem.** For any monotone submodular function with $f(\emptyset) = 0$, the greedy algorithm as described above (with k iterations) satisfies
$$f(S_k) \geq \left(1 - \frac{1}{e}\right) f(S_k^\star),$$
where $S_k^* = \arg\max_{S\,:\,|S|=k} f(S)$.
  - The proof of (a more general version of) the theorem is given below.
  - The factor $1 - \frac{1}{e}$ is *tight*; no algorithm using a *polynomial* number of function calls gets closer in general. (Proved by Nemhauser and Wolsey in 1978.)

- **An important question:** Is getting within $1 - 1/e \approx 0.63$ of the maximum "good enough"?
  - The answer: *It depends.* For a problem like coverage, if the optimal coverage is 100% and an algorithm only guarantees 63% coverage, it seems quite disappointing. But in an influence maximization problem, if influencing a million users would be optimal and we "only" manage to influence around 630,000, that 'eems pretty good.
  - In situations like the former, the more general theorem described below is useful.
  - Note also that 63% is only a worst-case guarantee, and the greedy algorithm often performs much better than the worst-case scenario in practice.

- We can in fact generalize this theorem by letting the size of the set returned (say ℓ) differs from the size of the set whose performance we are comparing against (say k):

- **Theorem.** For any monotone submodular function with $f(\emptyset) = 0$, the greedy algorithm modified to perform ℓ iterations (rather than k iterations) satisfies
$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S_k^\star)$$
For example, with $\ell = 5k$ iterations we are at least 99.3%-close to $f(S_k^\star)$ (so the set formed has size $5k$, but we are only guaranteeing its performance relative to the best size-k solution).
  - Proof outline (formalized below): Consider the "optimality gap" $f(S_k^*) - f(S_{\text{current}})$. By the greedy selection rule and submodularity, each item closes at least a $\frac{1}{k}$-fraction of gap. Hence, after ℓ iterations, the fraction is at most $\left(1 - \frac{1}{k}\right)^\ell \leq e^{-\ell/k}$.

- **(**Optional**)** There is also a well-known related result for the case that f is *integer-valued*, monotone, and $f(\emptyset) = 0$: To attain a desired target value q (often chosen as $q = f(V)$, the value of the entire set), the greedy algorithm uses at most a $1 + \log(\max_{v \in V} f(\{v\}))$ factor more function evaluations than the smallest possible. (See the early paper "An analysis of the greedy algorithm for the submodular set covering problem" for the case $q = f(V)$.)

– Here the goal is flipped – instead of "fix the number of evaluations and ask how high a value we can get", the problem is "fix the target value and ask how many evaluations we need to get there".

3.4 Proof Details

We prove the more general theorem. We abbreviate the optimal size-k solution, S_k^*, to simply S^*, and we denote its elements as $\{e_1^\star, e_2^\star, \cdots, e_k^\star\}$. Then for all iterations $i < \ell$, we have

$$
\begin{aligned}
f(S^\star) &\leq f(S^\star \cup S_i) && \text{by monotonicity} \\
&= f(S_i) + \sum_{j=1}^{k} \Delta\left(e_j^\star | S_i \cup \{e_1^\star, e_2^\star, \cdots, e_{j-1}^\star\}\right) && \text{by telescoping} \\
&\leq f(S_i) + \sum_{j=1}^{k} \Delta\left(e_j^\star | S_i\right) && \text{by submodularity} \\
&\leq f(S_i) + \sum_{j=1}^{k} \Delta\left(e_{i+1} | S_i\right) && \text{by def. of greedy selection rule} \\
&\leq f(S_i) + k\left(f(S_{i+1}) - f(S_i)\right). && \text{by } \sum_{j=1}^{k} c = kc \text{ and definition of } \Delta.
\end{aligned}
$$

Re-arranging gives $f(S^*) - f(S_{i+1}) \leq \left(1 - \frac{1}{k}\right)\left(f(S^*) - f(S_i)\right), \forall i < \ell$, and applying this recursively gives

$$
\begin{aligned}
f(S^\star) - f(S_\ell) &\leq \left(1 - \frac{1}{k}\right)^\ell f(S^\star) && \text{since } f(\emptyset) = 0 \\
&\leq e^{-\ell/k} f(S^*). && \text{since } 1 - x \leq e^{-x}, \forall x \in \mathbb{R}
\end{aligned}
$$

Rearranging terms yields $f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*)$.

4 Other Submodular Maximization Algorithms

The computational complexity of a standard implementation of the greedy algorithm is $O(nk)$, since in each of the k iterations a search is done over $O(n)$ items. This isn't particularly high complexity, but it could be desirable to reduce it in applications where n is large (e.g., image summarization on millions of images). The following subsections present two variations of the greedy algorithm that seek to address this.

4.1 Lazy Greedy

In an early paper (Minoux, 1978), it was observed that the number of function evaluations done by the greedy algorithm can be reduced (and in turn its computational complexity) without changing the algorithm's output. The idea is outlined as follows, and the algorithm is called Lazy Greedy:

- For each element $e \in V$, keep an upper bound $\rho(e)$ (initialized to ∞) on the marginal gain of adding e. The list of $(e, \rho(e))$ pairs is maintained and kept in decreasing order of $\rho(e)$.
- In a given iteration (say the i-th), evaluate the first element, say e (with the highest $\rho(e)$), and update $\rho(e) \leftarrow \Delta(e|S_{i-1})$. Let e' be the element just after e in the list, and consider the following two cases:

- If the updated $\rho(e)$ still satisfies $\rho(e) \geq \rho(e')$, then due to the list being sorted, we can conclude that it is the largest in the *entire* list. Hence, the greedy algorithm chooses e, it is removed from further consideration, and we proceed to iteration $i+1$.
- If the updated $\rho(e)$ instead satisfies $\rho(e) < \rho(e')$, then we insert e to the suitable later position in the list (to maintain the "decreasing order" property). After doing so, e' is at the start of the list, and we proceed back to the "...evaluate the first element..." step, repeating this process as needed until an element is selected.

Notice that in the first sub-case of Step (ii), although the later items in the list do not get updated, we still maintain the crucial property that $\rho(\cdot)$ is a valid upper bound. This is because, by submodularity, *any updates could only cause $\rho(\cdot)$ to get smaller*, but never larger.

It is difficult to theoretically quantify the precise computational savings of Lazy Greedy, but in practice it can run 10s, 100s, or even 1000s of times faster depending on the problem.

4.2 Stochastic Greedy

The Lazy Greedy algorithm is simply a more efficient way of implementing the greedy algorithm, giving the same answer. A randomized algorithm called Stochastic Greedy is a genuinely different algorithm that maintains the $1 - 1/e$ approximation guarantee but with much lower computation.

The modification is simple: At round i where S_{i-1} has already been chosen, choose N elements of $V \setminus S_{i-1}$ uniformly at random, and among those N, choose the one with the highest $\Delta(e|S_{i-1})$. (The greedy algorithm is similar but considers the entire set $V \setminus S_{i-1}$.)

The key idea is that the analysis for the greedy algorithm shows an improvement in iteration i of $(f(S^*) - f(S_{i-1}))/k$, and this improvement can be obtained by adding a uniformly random element of S^* to the current set (this can be shown using submodularity). Choosing $N = (n/k)\log(1/\epsilon)$ ensures overlapping with S^* with probability at least $1 - \epsilon$, which turns out to be sufficient. (This intuition is a bit imprecise because some elements of S^* might already be in S_{i-1}, but it gives the rough idea.)

Formalizing this idea (see the paper "Lazier than Lazy Greedy") gives the following guarantee: For any non-negative monotone submodular f, the Stochastic Greedy algorithm with $N = (n/k)\log(1/\epsilon)$ guarantees that the final set S_k satisfies

$$\mathbb{E}[f(S_k)] \geq (1 - 1/e - \epsilon) f(S_k^*),$$

where an expectation is included on the left-hand side in view of this being a randomized algorithm.

The runtime is now $O(k \cdot N) = O\big(n \log \frac{1}{\epsilon}\big)$, which is significantly smaller than $O(nk)$ if k is large.

4.3 (**Optional**) An Entirely Different Approach: Multilinear Extension

- Along similar lines to the above notes on submodular minimization, there are techniques for submodular maximizaiton (possibly constrained) based on extending the function from $\{0,1\}^n$ to $[0,1]^n$, applying a continuous optimization algorithm, and then converting back to $\{0,1\}^n$.
- For maximization, a different extension to $[0,1]^n$ turns out to be more useful (rather than the Lovász extension described above), called the *multilinear extension*.
- Again writing $f(S)$ and $f(\mathbf{x})$ interchangeably (where $\mathbf{x}$ has 1s at the entries indexed by S and 0s

elsewhere), the extension from $\{0,1\}^n$ to $[0,1]^n$ is as follows:

$$f(\mathbf{x}) = \mathbb{E}[f(S)]$$

where element i is included in S with probability x_i, with independence across i values.

- This function is not concave (which would be ideal for maximization), but it does satisfy $\frac{\partial^2 f}{\partial x_i \partial x_j} \leq 0$ for all (i,j), which ensures concavity along lines whose direction vector has all non-negative coefficients.
- Evaluating $f(\mathbf{x})$ exactly is hard, since there are exponentially many possible values of the random S. To circumvent this, one can take random samples from the distribution on S and calculate the average $f(\cdot)$ value with respect to those samples. Then standard concentration bounds (e.g., Hoeffding's inequality) can be used to show that the resulting estimate is accurate with high probability.

4.4 (**Optional**) Beyond Cardinality Constraints

The cardinality constraint $\{S : |S| \leq k\}$ is very common, but certainly not the only kind of constraint we might encounter when optimizing submodular functions. A significantly more general class if constraints is *matroid constraints*. Briefly, given some ground set V and a collection of subsets $\mathcal{I}$ (with each $S \in \mathcal{I}$ being a subset of V), we say that $(V, \mathcal{I})$ is a matroid if:

(i) If $S \in \mathcal{I}$ then any $S' \subset S$ is also in $\mathcal{I}$ (including $S' = \emptyset$);

(ii) If $A \in \mathcal{I}$, $B \in \mathcal{I}$, and $|B| > |A|$, then there exists $x \in B$ such that $A \cup \{x\} \in \mathcal{I}$.

In other words, (i) any removal of elements maintains feasibililty, and (ii) if one feasible set is larger than another, then we can move some element from the latter to the former while maintaining feasibility.

Some examples of matroids are as follows:

- The cardinality-constrained subset that we already studied is a matroid.
- V contains vectors in $\mathbb{R}^d$, and $\mathcal{I}$ consists of all linearly independent subsets.
- $G = (V, E)$ is a graph, and $\mathcal{I}$ contains all the subsets of edges that form a forest.

Returning to submodular maximization, we can now consider the problem

$$\max_{S \in \mathcal{I}} f(S), \tag{2}$$

where f is monotone, submodular, and $f(\emptyset) = 0$, and $(V, \mathcal{I})$ forms a matroid. Then, the standard greedy algorithm provides a $\frac{1}{2}$-approximation, and a more sophisticated algorithm admits a $\left(1 - \frac{1}{e}\right)$-approximation (e.g., see `https://chekuri.cs.illinois.edu/papers/submod_max.pdf`).

5 (**Optional**) Other Aspects of Discrete Optimization

This section gives a very brief overview of some important aspects of discrete optimization. For further detail, see my MA4254 lecture notes: `https://www.comp.nus.edu.sg/~scarlett/MA4254_LectureNotes.pdf`

- In the CS department, you can also consider **CS4234 Optimisation Algorithms**

5.1 Integer Linear Programming

Many problems in computer science and applied domains can be cast as an *integer linear program* (ILP):

$$\text{maximize}_{\mathbf{x}\in\mathbb{Z}^d} \quad \mathbf{c}^T\mathbf{x} \tag{3}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0} \tag{4}$$

for some matrix $\mathbf{A} \in \mathbb{R}^{m\times d}$ and vectors $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^d$, where inequalities between vectors (e.g., $\mathbf{x} \geq \mathbf{0}$) are taken element-wise (e.g., $x_i \geq 0$ for all $i = 1, \ldots, d$).

- The above form is called the *standard form*, and other forms can be converted to this form using some simple tricks. Another common form is that in which we have $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ instead of $\mathbf{A}\mathbf{x} = \mathbf{b}$.

It is especially common for $\mathbf{x}$ to be binary-valued, in which case the constraint $\mathbf{x} \in \{0,1\}^d\}$ (rather than $\mathbb{Z}^d$) may explicitly be included. Having linear constraints and a linear objective function may seem restrictive, but these are in fact very common (e.g., the cardinality constraint that we studied in this lecture is $\sum_{i=1}^n x_i \leq k$, which is linear).

Some examples of problems that can be cast as ILPs are as follows: Knapsack problem, matching problems, covering and packing problems, constraint satisfaction, facilities location problems, scheduling problems, traveling salesman problem (albeit with exponentially many constraints), etc.

5.2 LP Relaxations and Rounding

Solving ILPs is NP-hard in general, and a common approach is to try to relax the integer constraint $\mathbf{x} \in \mathbb{Z}^d$ to $\mathbf{x} \in \mathbb{R}^d$ to produce a *linear program* (LP):

$$\text{maximize}_{\mathbf{x}\in\mathbb{R}^d} \quad \mathbf{c}^T\mathbf{x} \tag{5}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0} \tag{6}$$

Similarly, in the binary case, the relaxation is from $\mathbf{x} \in \{0,1\}^d$ to $\mathbf{x} \in [0,1]^d$. Naturally, solving the LP will often give a solution with non-integer values, meaning some sort of *rounding procedure* is needed to produce something that is feasible for the ILP. This must be done with care, since naive rounding procedures may produce points that, while integer-valued, fail to satisfy $\mathbf{A}\mathbf{x} = \mathbf{b}$ (or $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ in the inequality case).

We briefly summarize the types of guarantees (if any) that might arise using an LP relaxation:

- *(Tight/exact)* If the LP solutions happens to already be integer-valued, then it is also optimal for the ILP. This may sound unrealistic, but it is guaranteed when $\mathbf{A}$ satisfies a property called *totally unimodular*, with examples including matching problems, minimum cut, and shortest path.

- *(Approximate)* After rounding, we might be able to guarantee that the solution is within some factor $\alpha \in (0,1)$ of optimal (for maximization problems), or is at most an $\alpha > 1$ factor higher than optimal (for minimization problems). This kind of guarantee is common in coverage problems (which are of the minimization type):

  - Vertex cover admits a 2-approximation.
  - More generally, set cover with every element being in at most K sets admits a K-approximation.

– Using a randomized rounding strategy, the general set cover problem admits an $O(\log n)$-approximation when there are n elements to be covered.

- *(Neither)* In some problems, even getting a constant-factor approximation is NP-hard. For example, solving the Traveling Salesman Problem to within a constant factor would imply solving the Hamiltonian Cycle problem, which is known to be NP-hard.

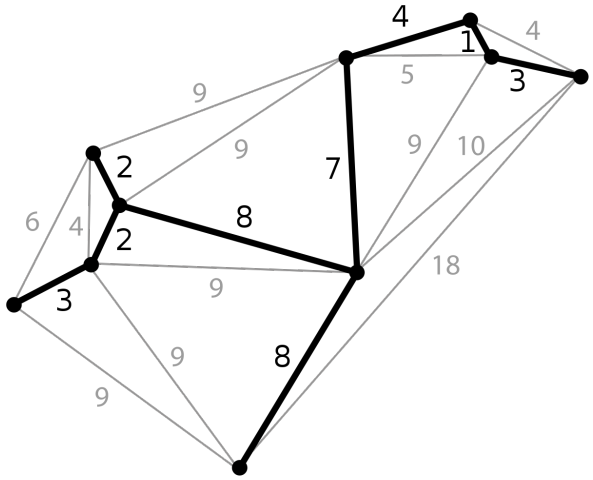
5.3 Greedy Algorithms

In Section 4.4, we discussed submodular maximization subject to a matroid constraint. It turns out that if the function is *modular* (i.e., linear), then we can say something much stronger, namely, a greedy algorithm gives the optimal solution. Concretely, the relevant class of optimization problems is

$$\max_{S \in \mathcal{I}} \sum_{i \in S} c_i, \quad \text{(or min instead of max)}$$

where $(V, \mathcal{I})$ forms a matroid, and $\{c_i\}_{i \in V}$ are fixed constants. Instead of giving a general statement, we state a well-known example of constructing a *minimum spanning tree*, which can be cast as above using the forest matroid that we mentioned earlier.

Let $G = (V, E)$ be an undirected graph with edge weights c_{ij}. A *tree* is a sub-graph for which every two nodes are connected by exactly one path (and hence there are no cycles). The *Minimum Spanning Tree* (MST) problem concerns selecting a subset of edges to form a spanning tree of minimum weight. An example is shown as follows:



The greedy algorithm is simply as follows: *Starting with the empty graph, repeatedly add the smallest-weight edge that does not form a cycle. Stop once there are $|V| - 1$ edges.* (Note: Every tree on n nodes has $n - 1$ edges.)

5.4 Global Optimization Methods

There are a number of global methods that guarantee finding the optimal solution for ILPs (and other problems beyond ILPs). While their runtime is exponential in the worst case, they can be very fast in practice. Briefly, some such algorithms include:

- *Branch and bound*, which is a divide-and-conquer approach that efficiently computes upper and lower bounds for various sub-problems (e.g., obtained by setting certain variables to 0 or 1), and using those bounds to rule out large parts of the search space.
- *Cutting plane methods*, which iteratively solve LP relaxations but then introduce further constraints that must hold in the ILP solution, while making the current LP relaxation solution infeasible.
- *Simulated annealing*, which iteratively performs updates with the goal of improving over time while avoiding local optima. Initially ("high temperature") most updates are accepted by the algorithm, but then over time ("cooling down") updates that provide improvements become more preferred, and eventually ("low temperature") only updates that provide improvements are accepted.

Appendix: Proving Submodularity

Exercise 1: *Let $c_1, \ldots, c_n$ be fixed non-negative numbers, and consider $S \subseteq \{1, \ldots, n\}$. Show that the function with $f(\emptyset) = 0$ and $f(S) = \max_{i \in S} c_i$ (when $S \neq \emptyset$) is submodular and non-decreasing.*

- Solution: The non-decreasing property is immediate from the fact that adding elements to S means taking the maximum over a larger set. To show submodularity, we consider $S \subseteq T$ and consider two cases:
  - If adding j to T doesn't increase the function for T, then we just use the fact that the increase is non-negative for S, and hence $\Delta(j|S) \geq \Delta(j|T)$ as required.
  - If adding j to T does increase the function, then it must be because j is the new maximum within $T \cup \{j\}$, and thus also the new maximum within $S \cup \{j\}$. Then by the non-decreasing property, the amount of increase is higher for S than for T.

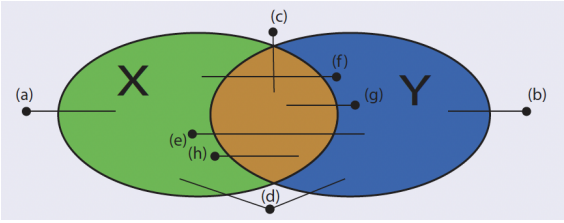
  You could also answer this question by comparing $f(A \cup B) + f(A \cap B)$ to $f(A) + f(B)$.

Exercise 2: *Given a graph $G = (V, E)$ and $S \subseteq V$, the cut function $f(S)$ is defined as the number of edges starting in S and ending in $V \setminus S$. Use a diagram to argue that*

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$$

which means that f is submodular. Furthermore, explain why f is non-monotone for any non-empty graph.

- Solution: Consider the categories of edges from the following figure:



  Let d_X and d_Y respectively be the contribution to (d) from X and from Y. Then, we have:
  - $f(X)$ is the sum of a, c, f, g, d_X

- $f(Y)$ is the sum of b, c, e, h, d_Y
- $f(X \cup Y)$ is the sum of a, b, c, d
- $f(X \cap Y)$ is the sum of c, g, h

Hence, $f(X) + f(Y)$ is $a + b + 2c + d + e + f + g + h$, and $f(X \cup Y) + f(X \cap Y) = a + b + 2c + d + g + h$, so the difference between the two is $e + f \geq 0$.

The function is non-monotone because $f(\emptyset) = f(V) = 0$, but as long as the graph is non empty there exists some S with $\emptyset \subset S \subset V$ such that $f(S) > 0$. So we add nodes starting from the empty set and ending with the full set, at some point the function increases and then decreases again.