Submodular Functions:
Theory, Exact Minimization, and Approximate Maximization

Authored by
Tony Oh, Parker Rho, Menghan Xu

This work introduces submodular functions through their equivalent definitions and the intuitions behind them, and examines how these properties lead to both tractable minimization algorithms and provably tight approximation algorithms for maximization, with an application to the Maximum Directed Cut problem.

§1 Definitions and Intuition of Submodular Functions

There are three equivalent definitions of submodular functions. While mathematically interchangeable, each definition provides a distinct intuition for understanding the concept.

Definition 1.1. (*Global*) Let N be a finite ground set and $f : 2^N \to \mathbb{R}$. Then f is submodular if for all $A, B \subseteq N$,

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

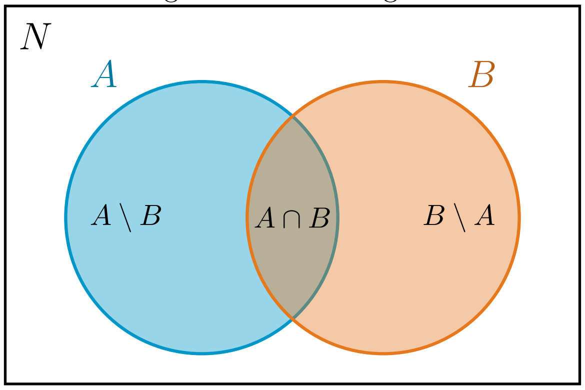
Definition 1.2. (*Marginal*) Denote $f_A(i) = f(A + i) - f(A)$. We say that f is submodular if for all $A \subseteq B \subseteq N$ and all $i \in N \setminus B$,

$$f_A(i) \geq f_B(i).$$

Definition 1.3. (*Local*) The function f is submodular if for all $A \subseteq N$ and all $i, j \in N \setminus A$,

$$f(A) - f(A + i) - f(A + j) + f(A + i + j) \leq 0.$$

Figure 1: Venn Diagram



To interpret **Definition** 1.1, it is helpful to rearrange the inequality as:

$$f(A \cup B) \leq f(A) + f(B) - f(A \cap B).$$

This form parallels the inclusion-exclusion principle visualized in the Venn diagram (**Figure** 1), where the area satisfies $|A \cup B| = |A| + |B| - |A \cap B|$. Indeed, if f were a *modular* function (which behaves linearly like cardinality or area), this relationship would hold with strict equality.

The submodularity condition ($\leq$), however, implies that the value of the union is strictly bounded by the sum of the components minus their intersection. Intuitively, this captures the concept of redundancy: the combined set $A \cup B$ yields less value than the simple sum of its parts because the overlapping information in $A \cap B$ provides no additional gain once merged. In other words, the whole is 'cheaper' than the sum of the parts.

To explain this in a more intuitive way, think of f as measuring satisfaction. Let A be a burger with fries, and B be pizza with fries. Their intersection, $A \cap B$, corresponds to the fries. When we compute $f(A) + f(B)$, the enjoyment from the fries is counted twice. However, in reality, eating a second portion of fries yields much less additional satisfaction due to satiety. As a result, the satisfaction from the combined meal, $f(A \cup B)$, is smaller than the sum of the satisfactions from the two meals considered separately.

While **Definition** 1.1 offers a global set-theoretic perspective, **Definition** 1.2 reframes submodularity through the lens of diminishing marginal returns. Since $A \subseteq B$, the set B represents a larger context (or endowment). The inequality implies that the incremental value of adding a new element i decreases as the underlying set grows larger. Intuitively, the more you already have, the less valuable a new addition becomes.

Rearranging the terms in **Definition** 1.3 yields:

$$f(A + i + j) - f(A + i) \leq f(A + j) - f(A).$$

This inequality not only echoes the principle of diminishing marginal returns (**Definition** 1.2) but also serves as a discrete analogue to concavity (corresponding to $f'' \leq 0$). Specifically, if we view the marginal gain as a discrete first derivative, this condition implies that the derivative is non-increasing as the input set grows, which mirrors the behavior of a concave function whose slope decreases.

However, this raises a deceptive question: does this concave-like property imply that maximizing a submodular function is computationally easy?

Although we do not want to spoil the punchline too early, this is exactly where the magic lies. The key takeaway is a striking paradox: while submodularity geometrically resembles concavity, it behaves computationally like convexity. Specifically, minimizing a submodular function is solvable in polynomial time (easy), whereas maximizing it is generally NP-hard. In the realm of optimization, submodular functions effectively act as discrete convex functions rather than concave ones.

§2 Submodular Minimization and Lovász Extension

This section aims to explain the computational connection between submodular functions and convex functions. As established by Lovász (1983), a discrete submodular function can be extended to a continuous convex function known as the Lovász extension. This fundamental link allows us to leverage tools from continuous convex optimization, most notably the Grötschel–Lovász–Schrijver (GLS) [4] algorithm (1988), to solve the submodular minimization problem

$$\min_{S \subseteq N} f(S)$$

in poly($|N|$) time.

A submodular function $f : 2^N \to \mathbb{R}$ can also be equivalently regarded as a function on the boolean hypercube $f : \{0,1\}^N \to \mathbb{R}$. This is simply because every subset S uniquely corresponds to an indicator vector $\mathbf{1}_S \in \{0,1\}^N$ (where the i-th entry is 1 if $i \in S$, and 0 otherwise).

The function has two canonical continuous extensions $f^+, f^- : [0,1]^N \to \mathbb{R}$, where f^+ is concave and f^- is convex.

Definition 2.1. For a function $f : \{0,1\}^N \to \mathbb{R}$, we define:

- The concave closure $f^+(\mathbf{x}) = \max \left\{ \sum_{S \subseteq N} \alpha_S f(S) : \sum_{S \subseteq N} \alpha_S \mathbf{1}_S = \mathbf{x}, \sum_{S \subseteq N} \alpha_S = 1, \alpha_S \geq 0 \right\}$.
- The convex closure $f^-(\mathbf{x}) = \min \left\{ \sum_{S \subseteq N} \alpha_S f(S) : \sum_{S \subseteq N} \alpha_S \mathbf{1}_S = \mathbf{x}, \sum_{S \subseteq N} \alpha_S = 1, \alpha_S \geq 0 \right\}$.

The Lovász extension is defined as follows.

Definition 2.2. (*Lovász extension*) For a function $f : \{0,1\}^N \to \mathbb{R}$, the Lovász extension $f^L : [0,1]^N \to \mathbb{R}$ is defined by:

$$f^L(\mathbf{x}) = \sum_{i=0}^{n} \lambda_i f(S_i)$$

where $\emptyset = S_0 \subset S_1 \subset S_2 \subset \cdots \subset S_n$ is a chain of subsets such that $\sum \lambda_i \mathbf{1}_{S_i} = \mathbf{x}$ and $\sum \lambda_i = 1$, $\lambda_i \geq 0$.

With these two extension definitions in place, we can now introduce the key theorem of this section.

Theorem 2.3. *The Lovász extension f^L and convex closure f^- are identical if and only if f is submodular.*

Proof. *(Sufficiency)* Let f be submodular. The value $f^-(\mathbf{x})$ is obtained by minimizing $\mathbb{E}[f(S)]$ over distributions representing $\mathbf{x}$. To resolve any ambiguity in the optimal distribution, we choose the one that maximizes the second moment of set sizes, $\sum \alpha_S |S|^2$. If the support of this distribution were not a chain, it would contain non-nested sets A, B with positive probability. By applying an "uncrossing" operation, moving weight from A, B to $A \cup B$, $A \cap B$, we maintain the vector constraint $\mathbf{x}$. Submodularity ensures the cost does not rise ($f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$), but the strictly convex nature of the function $h(k) = k^2$ ensures that $\sum \alpha_S |S|^2$ increases. This contradicts the maximality choice. Therefore, the support must be a chain, meaning the solution matches the greedy construction of the Lovász extension f^L.

(Necessity) Suppose f lacks submodularity. Then we can find a set S and items i, j where $f(S) + f(S+i+j) > f(S+i) + f(S+j)$. Let us evaluate both functions at $\mathbf{x} = \mathbf{1}_S + \frac{1}{2}(\mathbf{1}_i + \mathbf{1}_j)$. The Lovász extension effectively interpolates using the chain $S \subset S \cup \{i,j\}$, giving the average of $f(S)$ and $f(S+i+j)$. In contrast, the convex closure allows us to express $\mathbf{x}$ as an average of the indicators for $S \cup \{i\}$ and $S \cup \{j\}$. The non-submodularity condition implies that this second decomposition yields a strictly lower value than the Lovász calculation. Thus, $f^- \neq f^L$. □

This result implies that the convex closure of a submodular function can be evaluated efficiently. Since convex minimization is computationally tractable, this provides a theoretical justification for why submodular function minimization can be solved in polynomial time.

Theorem 2.4. *(Grötschel–Lovász–Schrijver) The problem $\min_{S \subseteq N} f(S)$ can be solved in polynomial time for any submodular function $f : 2^N \to \mathbb{R}$.*

The GLS algorithm uses the ellipsoid method. It relaxes the discrete problem into a continuous one and repeatedly shrinks an ellipsoid to find an optimal solution. Later, several faster combinatorial algorithms were developed, such as those by Iwata, Fleischer, and Fujishige, as well as by Schrijver. However, since we want to focus more on submodular maximization, we do not discuss these minimization algorithms further.

§3 Submodular Maximization and Greedy Algorithm

Now that we have established a solid understanding of the definition and intuition behind submodular functions, it is not hard to see that they are ubiquitous in combinatorial optimization. In fact, many natural phenomena fall into this category. For instance, coverage functions, entropy in information theory, and graph cut functions are all classic examples where the property of diminishing returns holds true.

As for the optimization of these functions, the landscape is divided. First, we have submodular minimization. We know that this can be solved in polynomial time, which makes it a powerful tool. In theory, this encompasses classic problems like the minimum cut in graphs and the minimum hypergraph cut. This theoretical tractability allows us to efficiently solve complex real-world problems such as image segmentation in computer vision and various clustering tasks.

Next, we turn to the most exciting part. This is submodular maximization. Unfortunately, this problem is generally NP-hard. This implies that, assuming P does not equal NP, no algorithm can find the exact global optimum in polynomial time. Since we cannot solve it efficiently, we rely on approximation algorithms to find high-quality solutions. Theoretical examples of this include maximum group coverage, the submodular welfare problem, and the maximum cut problem (which we will discuss in detail later). In practice, these models are applied to critical real-world tasks like sensor placement or data subset selection.

Formally, the general problem of submodular maximization can be formulated as:

$$\begin{aligned} \underset{S\subseteq N}{\text{maximize}} \quad & f(S) \\ \text{subject to} \quad & S \in \mathcal{I} \end{aligned}$$

The greedy algorithm is a natural heuristic. Although it can be applied to non-monotone submodular functions, it typically performs better for monotone functions, where $f(A) \leq f(B)$ whenever $A \subseteq B$.

Algorithm 1: Greedy Algorithm

```
1 $S \leftarrow \emptyset$, $A \leftarrow \emptyset$
2 repeat
3     $A \leftarrow \{e \mid S \cup \{e\} \in \mathcal{I}\}$
4     if $A \neq \emptyset$ then
5         $e \leftarrow \arg\max_{e' \in A} f_S(e')$
6         $S \leftarrow S \cup \{e\}$
7 until $A = \emptyset$;
8 return $S$
```

Theorem 3.1. *(Nemhauser–Wolsey–Fisher) The greedy algorithm achieves a $(1-\frac{1}{e})$-approximation for the problem*

$$\max_{|S|\leq k} f(S)$$

where $f : 2^N \to \mathbb{R}_+$ is a monotone submodular function.

Proof. Let O be a fixed optimal solution and let S_i be the partial solution of size i found by the greedy algorithm. The core of the proof lies in comparing the gain of the greedy choice to the "average" remaining value of the optimal set. At iteration $i+1$, the algorithm picks an element u maximizing the increment $f(S_i \cup \{u\}) - f(S_i)$. Due to submodularity, the total value added by the set O to the current set S_i is bounded by the sum of individual marginal gains:

$$f(O \cup S_i) - f(S_i) \leq \sum_{v \in O \setminus S_i} [f(S_i \cup \{v\}) - f(S_i)].$$

Because f is monotone, $f(O) - f(S_i) \leq f(O \cup S_i) - f(S_i)$. Combining these observations, the gap to optimality is bounded by a sum of at most k terms. Consequently, the maximum term (the greedy choice u) must cover at least a $1/k$ fraction of this sum:

$$f(S_{i+1}) - f(S_i) \geq \frac{1}{k}(f(O) - f(S_i)).$$

Rearranging terms reveals how the distance to the optimal value shrinks:

$$f(O) - f(S_{i+1}) \leq \left(1 - \frac{1}{k}\right)(f(O) - f(S_i)).$$

Starting with an initial gap of $f(O)$ and applying this reduction k times, the final gap is at most $(1 - 1/k)^k f(O)$. Using the standard exponential inequality, the final value satisfies $f(S_k) \geq (1 - 1/e)f(O)$.

□

Feige [2] proved that for this problem the approximation factor $(1 - \frac{1}{e})$ is optimal.

In other several settings, it provides good approximation ratios too, and until quite recently, these were the best known. Moreover, the simplicity of Greedy makes it useful in various applications where other complex algorithms may not be suitable.

§4 Improving the Greedy Algorithm for USM

The setting of Unconstrained Submodular Maximization (USM) is the most general form of submodular maximization. By "unconstrained", we mean that the input space over which we are searching for the maximum is defined as any possible subset of the ground set N where $|N| = n$. Aside from nonnegativity, there are also no additional properties, such as symmetry or monotonicity, that must hold for the function in question, f.

The greedy algorithm discussed in the previous section is known to perform well in constrained scenarios and for strictly monotone submodular functions [7], but fails for USM. [1] However, thanks to the work of Buchbinder et al. we also know of a rather simple linear time (1/2)-approximation for unconstrained submodular maximization (USM) [1]. We know that a (1/2)-approximation is in fact tight [3], meaning it has been proven to be hard to achieve an approximation of $(1/2 + \epsilon)$, so this algorithm is quite remarkable! The accomplishment lies in a slight adjustment to the straightforward greedy algorithm of **Section 3**, which alone achieves an approximation ratio of (1/3), but with a dash of randomness, they reached the coveted (1/2)-approximation. So, before we get into the tight approximation algorithm, let's first look at the (1/3)-approximation.

§4.1 The Deterministic (1/3)-Approximation

First, note that at each iteration of the straightforward greedy algorithm, it just looks at the set of feasibly addable elements, A, and picks the best one e that maximizes $f_S(e)$ (recall that this is the marginal value of adding e to S) starting with $A = \emptyset$. An equally reasonable algorithm is one that considers the compliment of f, $\overline{f}$, where $\overline{f}(S) := f(N \setminus S)$. Such an algorithm would start with $A = N$ and iteratively remove the best e at each iteration. Both of these are known to fail separately, but Buchbinder et al. [1] consider both options together, reaching the successful (1/3)-approximation in linear time:

Algorithm 2: Deterministic USM Greedy

```
1 $X_0 \leftarrow \emptyset$, $Y_0 \leftarrow N$
2 for $i = 1$ to $n$ do
3     $a_i \leftarrow f(X_{i-1} + e_i) - f(X_{i-1})$
4     $b_i \leftarrow f(Y_{i-1} - e_i) - f(Y_{i-1})$
5     if $a_i \geq b_i$ then
6         $X_i \leftarrow X_{i-1} + e_i, Y_i \leftarrow Y_{i-1}$
7     else
8         $X_i \leftarrow X_{i-1}, Y_i \leftarrow Y_{i-1} - e_i$
9 return $X_n$ (or equivalently $Y_n$)
```

The first thing you may notice is that since we are maximizing an unconstrained submodular function, there is no need to query an oracle to determine the feasibility of adding an element, so every element is considered for addition to X_i/removal from Y_i. The main technique this algorithm employs is considering the marginal gain between actions a_i, adding element e_i to X_i, and b_i, removing e_i from Y_i. Then the action with most marginal gain is selected. To prove the 1/3 approximation ratio, it is first useful to prove the following lemma:

Lemma 4.1. *For every $1 \leq i \leq n$*

$$a_i + b_i \geq 0.$$

Proof. We know for each element, it is either added to X_i for the first and only time and remains in Y_i or it is removed from Y_i for the first and only time and will never be added to X_i. Thus, $X_i \subseteq Y_i, \forall i$, implying $(X_{i-1} + e_i) \cup (Y_{i-1} - e_i) = Y_{i-1}$ and $(X_{i-1} + e_i) \cap (Y_{i-1} - e_i) = X_{i-1}$. This fact along with the global definition of submodularity (**Definition 1.1**) and some algebraic rearrangement shows

$$\begin{aligned} a_i + b_i &= [f(X_{i-1} + e_i) - f(X_{i-1})] + [f(Y_{i-1} - e_i) - f(Y_{i-1})] \\ &= [f(X_{i-1} + e_i) + f(Y_{i-1} - e_i)] - [f(X_{i-1}) + f(Y_{i-1})] \geq 0 \end{aligned}$$

□

Before the next step, we will introduce a new object, $OPT_i = (OPT \cup X_i) \cap Y_i$ which agrees with X_i and Y_i on the first i elements since $X_i \subseteq Y_i$ and with OPT on the last $n - i$ elements since those elements have not been processed yet and therefore cannot have been removed from Y_i. This new object allows us to directly quantify the marginal departure in value from OPT between iterations via $f(OPT_{i-1}) - f(OPT_i)$, or even the total distance from OPT at any iteration via $f(OPT_0) - f(OPT_i)$ since $OPT_0 = OPT$. The following lemma therefore seeks to bound $f(OPT_{i-1}) - f(OPT_i)$ by the total increase in value of both X_i and Y_i, $[f(X_i) - f(X_{i-1})] + [f(Y_i) - f(Y_{i-1})]$.

Lemma 4.2. *For every $1 \leq i \leq n$*

$$f(OPT_{i-1}) - f(OPT_i) \leq [f(X_i) - f(X_{i-1})] + [f(Y_i) - f(Y_{i-1})].$$

Proof. First assume that without loss of generality, $a_i \geq b_i$, meaning X gets updated by adding e_i and nothing happens to Y. In this case, $OPT_i = OPT_{i-1} + e_i$ and $f(Y_i) - f(Y_{i-1}) = 0$. Then the inequality in question becomes

$$f(OPT_{i-1}) - f(OPT_{i-1} + e_i) \leq f(X_i) - f(X_{i-1}) = a_i$$

There are now 2 cases, e_i was already $\in OPT$ and $e_i \notin OPT$.

Case 1 ($e_i \in OPT$)**:** The LHS of the inequality $= 0$ and since $a_i \geq b_i$ and $a_i + b_i \geq 0$ (by **Lemma 4.1**), the inequality clearly holds.

Case 2 ($e_i \notin OPT$)**:** We know that by definition, $OPT_{i-1} \subseteq Y_{i-1}$. Since in this case $e_i \notin OPT_{i-1}$, we further know that $OPT_{i-1} \subseteq Y_{i-1} - e_i$. Now we can use the marginal definition of submodularity (**Definition 1.2**):

$$\begin{gathered} f_{Y_{i-1}-e_i}(e_i) \leq f_{OPT_{i-1}}(e_i) \\ f(Y_{i-1}) - f(Y_{i-1} - e_i) \leq f(OPT_{i-1} + e_i) - f(OPT_{i-1}) \\ f(OPT_{i-1}) - f(OPT_{i-1} + e_i) \leq f(Y_{i-1} - e_i) - f(Y_{i-1}) = b_i \leq a_i \end{gathered}$$

□

Now we have all the tools to show that **Algorithm 2** is a (1/3)-approximation of the Unconstrained Submodular Maximization problem! If we sum up the inequality of **Lemma 4.2**, we get telescopic sums on both sides which we can collapse

$$\begin{aligned} \sum_{i=1}^{n} f(OPT_{i-1}) - f(OPT_i) &\leq \sum_{i=1}^{n}[f(X_i) - f(X_{i-1})] + \sum_{i=1}^{n}[f(Y_i) - f(Y_{i-1})] \\ f(OPT_0) - f(OPT_n) &\leq [f(X_n) - f(X_0)] + [f(Y_n) - f(Y_0)] \\ &\leq f(X_n) + f(Y_n) \\ f(OPT) - f(X_n) &\leq f(X_n) + f(X_n) \end{aligned}$$

The last line follows because it was previously mentioned that $OPT_0 = OPT$ and by definition $OPT_n = X_n = Y_n$. Then we can simplify the above inequality to say that $f(OPT)/3 \leq f(X_n) = f(Y_n)$.

§4.2 Max-cut Example of Deterministic Greedy

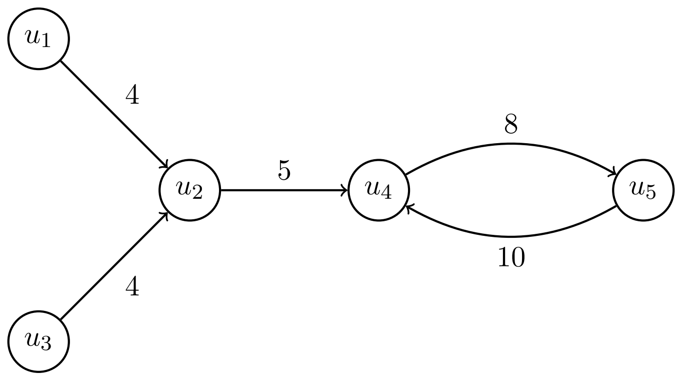
Now that we have proved the correctness of **Algorithm 2**, we will show an example of it being used in solving a specific instance of the Maximum Directed Cut problem. Consider a directed graph $G = (V, E)$ where $V = \{u_1, u_2, u_3, u_4, u_5\}$ and edges with weights $w(u_1, u_2) = 4$, $w(u_3, u_2) = 4$, $w(u_2, u_4) = 5$, $w(u_4, u_5) = 8$, and $w(u_5, u_4) = 10$.

The goal of the Maximum Directed Cut problem is to find a subset of vertices S that maximizes the weight of edges directed from S to $V \backslash S$. We start by initializing $X_0 \leftarrow \emptyset$, $Y_0 \leftarrow V$. We then execute the algorithm looking at one node (element) at a time, calculating and comparing each a_i and b_i. The function f here is the function of the cut value.

$$f(S) = \sum_{u \in S, v \notin S} w(u, v)$$

1. We start with u_1.

$$\begin{aligned} a_1 &\leftarrow f(X_0 + u_1) - f(X_0) = w(u_1, u_2) - 0 = 4 \\ b_1 &\leftarrow f(Y_0 - u_1) - f(Y_0) = 0 - 0 = 0 \end{aligned}$$



Because $a_1 \geq b_1$, $X_1 \leftarrow X_0 + \{u_1\}$ and $Y_1 \leftarrow Y_0$. At the end of iteration 1, we have the following.

(a) $X_1 = \{u_1\}$

(b) $Y_1 = \{u_1, u_2, u_3, u_4, u_5\}$

2. For u_2, if we add u_2 to X, we gain the edge (u_2, u_4). If we remove u_2 from Y, we lose edges (u_1, u_2) and (u_3, u_2).

$$\begin{aligned} a_2 &\leftarrow f(X_1 + u_2) - f(X_1) = w(u_2, u_4) - w(u_1, u_2) = 5 - 4 = 1 \\ b_2 &\leftarrow f(Y_1 - u_2) - f(Y_1) = w(u_1, u_2) + w(u_3, u_2) - 0 = (4 + 4) - 0 = 8 \end{aligned}$$

Because $a_2 \leq b_2$, $X_2 \leftarrow X_1$ and $Y_2 \leftarrow Y_1 - u_2$. Notice how the algorithm considers both the consequences of gaining the edges leaving u_2 and losing the edges coming into u_2. This is what makes our algorithm successful in achieving a good approximation ratio.

(a) $X_2 = \{u_1\}$

(b) $Y_2 = \{u_1, u_3, u_4, u_5\}$

3. The decision is rather simple for u_3 because it follows that of u_1 closely.

$$\begin{aligned} a_3 &\leftarrow f(X_2 + u_3) - f(X_2) = (w(u_1, u_2) + w(u_3, u_2)) - 0 = (4 + 4) - 4 = 4 \\ b_3 &\leftarrow f(Y_2 - u_3) - f(Y_2) = w(u_1, u_2) - (w(u_1, u_2) + w(u_3, u_2)) = 4 - (4 + 4) = -4 \end{aligned}$$

Because $a_3 \geq b_3$, $X_3 \leftarrow X_2 + \{u_3\}$ and $Y_3 \leftarrow Y_2$.

(a) $X_3 = \{u_1, u_3\}$

(b) $Y_3 = \{u_1, u_3, u_4, u_5\}$

4. For u_4, there are three edges to consider going in and out of u_4.

$$\begin{aligned} a_4 &\leftarrow f(X_3 + u_4) - f(X_3) = (4 + 4 + 8) - (4 + 4) = 8 \\ b_4 &\leftarrow f(Y_3 - u_4) - f(Y_3) = (4 + 4 + 10) - (4 + 4) = 10 \end{aligned}$$

Because $a_4 \leq b_4$, $X_4 \leftarrow X_3$ and $Y_4 \leftarrow Y_3 - u_4$. This is another case where the algorithm makes a good decision because we see that we don't take u_4 greedily for the edge weight of 8 for (u_4, u_5). We consider both cases of including and excluding u_4 before making the decision. While the choice is still greedy, we consider the locally optimum choice.

   (a) $X_4 = \{u_1, u_3\}$
   (b) $Y_4 = \{u_1, u_3, u_5\}$

5. Node u_5 is last.

$$a_5 \leftarrow f(X_4 + u_5) - f(X_4) = (4 + 4 + 10) - (4 + 4) = 10$$
$$b_5 \leftarrow f(Y_4 - u_5) - f(Y_4) = (4 + 4) - (18) = -14$$

   Because $a_5 \geq b_5$, $X_5 \leftarrow X_4 + \{u_5\}$ and $Y_5 \leftarrow Y_4$.

   (a) $X_5 = \{u_1, u_3, u_5\}$
   (b) $Y_5 = \{u_1, u_3, u_5\}$

   We see that our algorithm reached a final cut value of $f(X_5) = w(u_1, u_2) + w(u_3, u_2) + w(u_5, u_4) = 4 + 4 + 10 = 18$, which is the maximum cut value in this directed graph. Note that while this instance provided an optimal solution, **Algorithm 2** provides a (1/3)-approximation of the Maximum Directed Cut problem and other unconstrained submodular maximization problems.

§4.3 The Randomized (1/2)-Approximation

In this section, we will present a better randomized algorithm that attains a (1/2)-approximation on USM. The algorithm is very similar to the deterministic algorithm. The main difference is that instead of making greedy deterministic choices between a_i and b_i, we take a random choice with probability proportional to $\frac{a_i}{b_i}$. That is, we take add e_i to X if $a_i \geq b_i$ or remove e_i from Y otherwise. This smoothing of our choices will lead to a better approximation ratio which we will prove later.

Algorithm 3: Randomized USM

```
1  $X_0 \leftarrow \emptyset$, $Y_0 \leftarrow N$
2  for $i = 1$ to $n$ do
3      $a_i \leftarrow f(X_{i-1} + e_i) - f(X_{i-1})$
4      $b_i \leftarrow f(Y_{i-1} - e_i) - f(Y_{i-1})$
5      $a'_i \leftarrow \max\{a_i, 0\}, b'_i \leftarrow \max\{b_i, 0\}$
6      if $a'_i + b'_i > 0$ then
7          $p'_i \leftarrow \frac{a'_i}{a'_i + b'_i}$
8      else
9          $p'_i \leftarrow 1$
10     with probability $p'_i$, do $X_i \leftarrow X_{i-1} + e_i, Y_i \leftarrow Y_{i-1}$
11     else $X_i \leftarrow X_{i-1}, Y_i \leftarrow Y_{i-1} - e_i$
12 return $X_n$ (or equivalently $Y_n$)
```

The proof to show that this algorithm achieves a (1/2)-approximation is quite similar to the one for the deterministic algorithm. The main difference lies in the fact that all X_i and Y_i are now random variables, making all OPT_i random variables as well. So, the equivalent of **Lemma 4.2** is

Lemma 4.3. *For every $1 \leq i \leq n$,*

$$\mathbb{E}[f(OPT_{i-1}) - f(OPT_i)] \leq \frac{1}{2}\mathbb{E}[f(X_i) - f(X_{i-1}) + f(Y_i) - f(Y_{i-1})].$$

Before diving into the proof, Buchbinder et al. observe that it is sufficient to prove **Lemma 4.3** conditioned on any event where $X_{i-1} = S_{i-1}$ where $S_{i-1} \subseteq \{e_1, \dots, e_{i-1}\}$ that has nonzero probability. Now, for any fixed constant of X_{i-1}, we can see that Y_{i-1}, OPT_{i-1}, a_i, and b_i are by definition reliant on X_{i-1} and likewise will be constants and not random variables which makes things convenient for analysis. This event fixing is sufficient because of the definition of expectation being a probability weighted sum of outcomes. So, if the inequality holds for any fixed constant S_{i-1} with nonzero probability of occurring, then we can conclude the lemma. So for the proof, we will assess the inequality in question conditioned under the aforementioned event.

Proof. Since **Lemma 4.1** still holds due to a_i and b_i being constants, a_i and b_i will never both be strictly less than zero. Then it is sufficient to consider three cases where two of the three are analogous.

Case 1 ($a_i \geq 0$ and $b_i \leq 0$): It must be that $a'_i/(a'_i + b'_i) = 1$ so $Y_i = Y_{i-1}$, $f(Y_i) - f(Y_{i-1}) = 0$, and $X_i \leftarrow S_{i-1} + e_i$. So, similar to the proof for the deterministic algorithm, the inequality in question becomes

$$f(OPT_{i-1}) - f(OPT_i) \leq \frac{1}{2}[f(X_i) - f(X_{i-1})] = \frac{a_i}{2}$$

Once again when we consider the case where $e_i \in OPT$, it is clear that the inequality holds since the LHS is 0 while the RHS is nonnegative. If $e_i \notin OPT$, then we just use the same argument using the marginal definition of submodularity as in the proof for the deterministic algorithm to see that

$$\begin{aligned} f(OPT_{i-1}) - f(OPT_i) &\leq f(Y_i - e_i) - f(Y_{i-1}) = b_i \\ &\leq 0 \\ &\leq \frac{a_i}{2} \end{aligned}$$

Case 2 ($a_i < 0$ and $b_i \geq 0$): Notice now that $b'_i/(a'_i + b'_i) = 1$ and so we can just use the same technique as in the previous case.

Case 3 ($a_i \geq 0$ and $b_i > 0$): This is where we truly depart from an analysis that mirrors that of the one from **Subsection 4.1**. In this case, $a'_i = a_i$ and $b'_i = b_i$ so

$$\begin{aligned} \mathbb{E}[f(X_i) - f(X_{i-1}) + f(Y_i) - f(Y_{i-1})] &= \frac{a_i}{a_i + b_i}[f(X_{i-1} + e_i) - f(X_{i-1})] \\ &\quad + \frac{b_i}{a_i + b_i}[f(Y_{i-1} - e_i) - f(Y_{i-1})] \\ &= \frac{a_i}{a_i + b_i} \cdot a_i + \frac{b_i}{a_i + b_i} \cdot b_i \\ &= \frac{a_i^2 + b_i^2}{a_i + b_i}. \end{aligned}$$

Then:

$$\begin{aligned} (*) = \mathbb{E}[f(OPT_{i-1}) - f(OPT_i)] = &\frac{a_i}{a_i + b_i}[f(OPT_{i-1}) - f(OPT_{i-1} + e_i)] \\ &+ \frac{b_i}{a_i + b_i}[f(OPT_{i-1}) - f(OPT_{i-1} - e_i)] \end{aligned}$$

which doesn't look like much right now, but first notice that only one of these terms can be nonzero at a time since either $e_i \in OPT$ or $e_i \notin OPT$. Then we can go back to old faithful:

the marginal definition of submodularity! When $e_i \in OPT$, the first term of (*) equals zero and $X_{i-1} \subseteq OPT_{i-1} - e_i$ since $X_{i-1} \cap Y_{i-1} = X_{i-1}$ and e_i could not be in X_{i-1}, meaning

$$f(OPT_{i-1}) - f(OPT_{i-1} - e_i) \le f(X_{i-1} + e_i) - f(X_{i-1}) = a_i.$$

Then when $e_i \notin OPT$, the second term of (*) equals zero and we know from before that

$$f(OPT_{i-1}) - f(OPT_i) \le f(Y_i - e_i) - f(Y_{i-1}) = b_i.$$

So in both cases,

$$(*) \le \frac{a_i b_i}{a_i + b_i}$$

Hence, we can conclude this proof by showing that

$$\begin{aligned} \frac{a_i b_i}{a_i + b_i} &\le \frac{1}{2} \cdot \frac{a_i^2 + b_i^2}{a_i + b_i} \\ \frac{2a_i b_i}{a_i + b_i} &\le \frac{a_i^2 + b_i^2}{a_i + b_i} \\ 0 &\le \frac{a_i^2 - 2a_i b_i + b_i^2}{a_i + b_i} \\ &= \frac{1}{a_i + b_i}(a_i - b_i)^2 \end{aligned}$$

which is clearly true since $a_i + b_i \ge 0$ by **Lemma 4.1**. □

Now, like the deterministic algorithm, we can use **Lemma 4.3** by creating a telescopic sum across all $1 \le i \le n$ to get

$$\begin{aligned} \mathbb{E}[f(OPT_0) - f(OPT_n)] &\le \frac{1}{2}\mathbb{E}[f(X_n) - f(X_0) + f(Y_n) - f(Y_0)] \\ &\le \frac{\mathbb{E}[f(X_n) + f(Y_n)]}{2} \\ f(OPT)/2 &\le \mathbb{E}[f(X_n)] = \mathbb{E}[f(Y_n)], \end{aligned}$$

showing that this algorithm is a (1/2)-approximation in expectation!

§5 The End

We hope you enjoyed learning about submodular functions! :)

References

[1] Niv Buchbinder, Moran Feldman, Joseph SeffiNaor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM Journal on Computing*, 44(5):1384–1402, 2015.

[2] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[3] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

[4] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

[5] Jan Vondrák. Continuous extensions of submodular functions. Lecture notes, CS 369P: Polyhedral Techniques in Combinatorial Optimization, Stanford University, November 2010. Lecture 17.

[6] Jan Vondrák. The submodular allocation problem. Lecture notes, CS 369P: Polyhedral Techniques in Combinatorial Optimization, Stanford University, December 2010. Lecture 18.

[7] Jan Vondrák. Submodular functions. Lecture notes, CS 369P: Polyhedral Techniques in Combinatorial Optimization, Stanford University, November 2010. Lecture 16.

[8] Jan Vondrák. Submodular maximization subject to a matroid constraint. Lecture notes, CS 369P: Polyhedral Techniques in Combinatorial Optimization, Stanford University, December 2010. Lecture 19.