Greedy Maximization of Submodular Functions

David Rolnick & Jonathan Weed

December 12, 2014

1 Introduction

Traditional optimization techniques often rely upon functions that are convex or at least locally convex. Such diverse methods as gradient descent, loopy belief propagation, and linear programming all rely upon convex functions. However, many natural functions are not convex, yet optimizing over them is both possible and necessary. The class of *submodular functions* is particularly well-behaved and applicable. Submodularity may be thought of as a discrete analogue of convexity for set-functions (that is, functions defined on sets).

Minimizing convex functions is generally easy, while maximizing them is generally computationally hard. The same is true of submodular functions, which admit strongly polynomial-time minimization algorithms but which cannot generally be maximized efficiently. An example of this phenomenon, which we explore in some detail below, is the well-known difference between the problems of finding a minimum cut and finding a maximum cut in a graph. Minimum cuts can be found by any efficient maximum-flow algorithm, whereas finding a maximum cut is generally NP-hard.

In this paper, we describe two "greedy" approaches to the problem of submodular maximization. As we will show below, maximizing a submodular function is provably hard in a strong sense; nevertheless, simple greedy algorithms provide approximations to optimal solutions in many cases of practical significance. We first restrict ourselves to the problem of maximizing a *monotone* submodular function subject to a natural constraint. Here, a naïve greedy algorithm finds a good approximation but is inefficient. We present a recent improvement of the greedy algorithm with a much better asymptotic running time due to Badanidiyuru and Vondrák [1]. We then consider general submodular maximization. A naïve greedy algorithm is ineffective in this case, but we will present a new algorithm due to Buchbinder et al. [2] that combines two greedy algorithms to achieve a (1/2)-approximation guarantee. We end by considering open problems in this area and briefly surveying some recent trends in the field.

1.1 Preliminaries

A set-function $f : 2^X \to \mathbb{R}^+$ is said to be *submodular* if for every $A, B \subseteq X$, the following property holds:

$$f(A \cup B) + f(A \cap B) \leq f(A) + f(B). \tag{1.1}$$

For convenience, we also assume that $f(\emptyset) = 0$.

For any subset $A \subseteq X$, we can define a "marginal" function $f_A : 2^X \to \mathbb{R}^+$ by the rule $f_A(x) = f(A \cup \{x\}) - f(A)$. Then Equation (1.1) implies the following alternate characterization: a function f is submodular if

$$f_A(x) \leq f_B(x) \qquad \text{for all } B \subset A \text{ and } x \notin A. \tag{1.2}$$

In fact, Equation (1.2) is equivalent to Equation (1.1) for the purposes of defining submodularity. Equation (1.2) may be thought of a "law of diminishing returns": As a set grows, the advantage of adding new elements drops. Accordingly submodular functions have applications in economics, as well as mathematics, physics, and engineering.

In many case, submodular functions arise because a problem secretly involves matroids. A *matroid* $\mathcal{M}$ is a pair $(X, \mathcal{I})$, where X is a ground set and $\mathcal{I}$ a collection of subsets of X, with the following properties:

1. the empty set is contained in $\mathcal{I}$,
2. if $A \in \mathcal{I}$ and $B \subset A$ then $B \in \mathcal{I}$,
3. (Exchange Property) if $A \in \mathcal{I}$ and $B \in \mathcal{I}$ with $|A| > |B|$, then there exists $a \in A \setminus B$ such that $B \cup \{a\} \in \mathcal{I}$.

The elements of $\mathcal{I}$ are called *independent sets* by analogy to independent sets of vectors, which indeed motivate the definition of a matroid. Note that the Exchange Property holds for independent sets of vectors, since any independent set may be enlarged until its cardinality equals the dimension of the space, which is indeed the maximum cardinality across independent sets. The Exchange Property implies that all maximal independent sets have the same size, which is called the *rank* of the matroid.

Let $\mathcal{M}$ be a matroid over some ground set X. The *rank function* r of $\mathcal{M}$ is defined over all subsets $S \subseteq X$ by the rule that $r(S) = \max(|A| \mid A \in \mathcal{I}, A \subseteq S)$ equals the maximum cardinality of any independent set contained in S. Thus the rank of the matroid $\mathcal{M}$ equals $r(X)$. It may readily be shown, using the Exchange Property, that the rank function of a matroid is submodular, and this is in some ways the canonical example of a submodular function.

1.2 Examples of submodular functions

Directed cuts

Suppose we are given a directed graph $G = (V, E)$. For each subset $A \subseteq V$, define $f(A)$ to be the number of edges running from A to $V \backslash A$. We claim that f is a submodular function. Consider two subsets $A, B \subseteq V$, and refer to $f(A \cup B) + f(A \cap B)$ and $f(A) + f(B)$ as LHS and RHS, respectively. We must then prove that LHS $\leq$ RHS.

Consider an edge $e \in E$. If the source of e is in neither A nor B, then e is not counted in LHS or RHS. If e runs from A to B or vice versa, then it is counted once in $f(A)$ or $f(B)$ and at most once in $f(A \cap B)$; it therefore increases RHS at least as much as LHS. If e runs from A or B to $V \backslash (A \cup B)$, then e is counted twice in LHS and RHS if the head of e is in $A \cap B$ and is otherwise counted once in LHS and RHS. We conclude that LHS $\leq$ RHS and so f is indeed submodular.

Facility location

Recall the problem of facility location, which was considered during this course. We are given a set X of facilities and a set Y of clients. We wish to build some number of facilities and connect each client to exactly one of the facilities that has been opened. For every $i \in X$, there exists a cost f_i associated with opening facility i. For each facility-client pair (i, j), where $i \in X$ and $j \in Y$, there exists a benefit c_{ij} associated with pairing client j with facility i.

Define a function $f : 2^X \to \mathbb{R}^+$ by setting $f(A) = \sum_{i \in A} f_i$ for each $A \subseteq X$. Thus, f gives the total cost of opening a set of facilities. Also define a function $g : 2^X \to \mathbb{R}^+$ by setting $g(A) = \sum_{j \in Y} \max_{i \in A} c_{ij}$. Thus, g gives the total benefit of opening a set of facilities. We wish to maximize $g - f$; we claim that this function is submodular.

Observe first that $-f$ is submodular. This is because, for every $A, B \subseteq X$, we have

$$-f(A \cup B) - f(A \cap B) = -f(A) - f(B).$$

Now, we show that g is submodular. Let $g_j(A) = \max_{i \in A} c_{ij}$. Observe that

$$\begin{aligned} g_j(A \cup B) &= \max(g_j(A), g_j(B)) \\ g_j(A \cap B) &\leq \min(g_j(A), g_j(B)), \end{aligned}$$

from which it follows that g_j is submodular. It is simple to verify that the sum of submodular functions is itself submodular; therefore, $g = \sum_j g_j$ is submodular, as is $g - f$. The problem of facility location thus reduces to maximizing the submodular function $g - f$.

1.3 Minimizing a submodular function

We remarked above that minimizing a submodular function is computationally tractable. One way to find a minimum is to extend a submodular function to a continuous function defined on the unit hypercube. This extension, due to Lovász, has two properties that make it particularly useful, both of which are established in [7].

First, it is convex. Finding the minimum of a convex function over the hypercube is straightforward; an algorithm is given in Grötschel, Lovász, and

Schrijver [5]. This algorithm proceeds by a version of the ellipsoid method and therefore takes polynomial time, though it is slow to implement in practice.

Second, the extension achieves its minima at the vertices of the hypercube, so optimizing the Lovász extension yields an optimum of the original submodular function automatically, without the use of any rounding procedure.

The Lovász extension therefore gives a polynomial-time algorithm for minimizing submodular functions. In fact, there are purely combinatorial algorithms, which, while somewhat more complicated, solve the problem in strongly polynomial time.

2 Maximizing monotone submodular functions subject to a cardinality constraint

Before turning to the general problem of submodular maximization, we will focus on a special case. In this section, we consider a natural class of submodular functions: those that are *monotone*—that is, functions f for which $f(A) \leq f(B)$ whenever $A \subseteq B$. It is clearly trivial to maximize a monotone submodular function that is defined on all subsets of the ground set X; we simply take the entirety of X. Therefore, we suppose that our submodular function is defined only on certain subsets of X. The simplest such constraint is a cardinality constraint, where we restrict our attention to subsets of size at most k, and we consider this case here.

The problem of maximizing a monotone submodular function under such a constraint is still NP-hard since it captures such well-known NP-hard problems as Minimum Vertex Cover, in which we are given an undirected graph $G = (V, E)$ and must find a minimum-cardinality set of vertices such that every edge is incident to at least one of our vertices. To see this, first note that Minimum Vertex Cover reduces to determining, for each k, whether there exists a set of at most k vertices that forms a vertex cover.

Define a function $f : 2^V \to \mathbb{R}^+$ by setting $f(A)$ equal to the total number of edges incident to vertices in A. Now, f is submodular, because for $A, B \in 2^E$, $f(A) + f(B)$ equals the number $f(A \cup B)$ of edges incident to A or B, plus the number of edges incident to both A and B—this latter number is at least $f(A \cap B)$. Thus, there exists a vertex cover with at most k vertices if and only if $|E|$ is the maximum of f over subsets of size at most k. We conclude that it is NP-hard to maximize f subject to this cardinality constraint.

A simple greedy algorithm nevertheless gives a $(1 - 1/e)$-approximation to the problem of maximizing a monotone submodular function subject to a cardinality constraint. We begin by analyzing the algorithm and then show how to implement it efficiently using a "threshold algorithm."

2.1 The greedy algorithm

The greedy algorithm finds a solution $B \subset X$ in successive stages by selecting the element $x \in X$ at each stage that increases f the most. More formally, if B_i

is the subset selected after the ith step, then we let $B_{i+1} = B_i \cup \{x_{i+1}\}$, where $x_{i+1} \notin B_i$ maximizes the marginal function $f_{B_i}(x) = f(B_i \cup \{x\}) - f(B_i)$.

Equation (1.2) implies that the marginal values $f_\emptyset(x_1), f_{B_1}(x_2), \ldots$ form a nonincreasing sequence. Proving that the greedy algorithm provides the claimed approximation involves showing that this sequence does not decrease too quickly—in fact, that it is bounded below by a constant factor times the difference between the current solution and the optimum.

Proposition 2.1 (Nemhauser et al. [9]). *Let $(B_i)_{i=0}^k$ be a sequence of sets defined greedily as above, and let O be the set of size k such that $f(O)$ is as large as possible. Then*

$$f(B_k) \geq (1 - 1/e) f(O). \tag{2.1}$$

Proof. We will first show that the marginal increase at every stage is not too small. Our goal will be to establish that

$$f(B_{i+1}) - f(B_i) \geq \frac{1}{k}(f(O) - f(B_i)) \tag{2.2}$$

for all nonnegative $i \leq k$. The $f(B_{i+1}) - f(B_i)$ on the left-hand side of Equation (2.2) is the increase corresponding to the addition of element x_{i+1} to the set. Equation (2.2) establishes that if this increase is small, then the value of $f(B_i)$ is close to optimal.

The proof of Equation (2.2) relies on the following idea: whenever we choose not to include an element of O in some set B_i, it is because the marginal value of the elements of O is small, so $f(O)$ cannot be much larger than $f(B_i)$.

We make this rigorous as follows. We arbitrarily label the elements of $O = \{o_1, \ldots, o_k\}$ and consider adding them one at a time. For all i, we have by monotonicity

$$f(O) \leq f(O \cup B_i) = f(B_i) + \sum_{j=1}^{k} \left[f(B_i \cup \{o_1, \ldots, o_j\}) - f(B_i \cup \{o_1, \ldots, o_{j-1}\}) \right].$$

We can write each term $f(B_i \cup \{o_1, \ldots, o_j\}) - f(B_i \cup \{o_1, \ldots, o_{j-1}\})$ as $f_{B_i \cup \{o_1, \ldots, o_{j-1}\}}(o_j)$. Now $f_{B_i \cup \{o_1, \ldots, o_{j-1}\}}(o_j) \leq f_{B_i}(o_j)$ by Equation (1.2). So

$$f(O) \leq f(B_i) + \sum_{j=1}^{k} f_{B_i}(o_j) \leq f(B_i) + k f_{B_i}(x_{i+1}),$$

since x_{i+1} is the element of highest marginal value. Then Equation (2.2) follows.

To prove Equation (2.1), we rewrite Equation (2.2) as

$$[f(O) - f(B_i)] - [f(O) - f(B_{i+1})] \geq \frac{1}{k}(f(O) - f(B_i))$$

and rearrange to yield

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

Since $f(B_O) = f(\emptyset) = 0$, induction on i yields

$$f(O) - f(B_k) \leq \left(1 - \frac{1}{k}\right)^k f(O) \leq e^{-1} f(O),$$

and Equation (2.1) follows. □

Proposition 2.1 first appeared in a paper by Nemhauser, Wolsey, and Fisher, and Nemhauser and Wolsey subsequently showed in [8] that the approximation ratio $(1 - 1/e)$ is optimal if the algorithm can only evaluate f on polynomially many inputs.

2.2 Implementing the greedy algorithm

The greedy algorithm is both conceptually simple and provably optimal, which makes it a natural candidate for use in practice. A naïve implementation requires $O(kn)$ time, since one element is added to B at each step and each step requires checking the function on n different inputs. In [1], Badanidiyuru and Vondrák show how a lazy threshold algorithm can achieve a $(1 - 1/e - \epsilon)$-approximation in $O(\frac{n}{\epsilon} \log \frac{n}{\epsilon})$ time. This algorithm is also simple to implement and offers significant speedup when k is large.

The threshold algorithm is so named because at each stage it adds all elements that are "good enough," that is, whose marginal values are greater than a certain threshold. We noted earlier that the marginal values of new elements decrease monotonically as the greedy algorithm progresses; likewise, the threshold in the lazy implementation continues to decrease until a predetermined lower bound is reached. Even though the choices at each step are no longer optimal, we can still recover an almost optimal approximation.

The speedup arises for two reasons. The first is that we add multiple elements at each step, so we do not need k iterations to build the set B. The second is that, as in many approximation algorithms, we ignore elements whose contribution is so small that they will not affect the solution up to a factor of ϵ. In our case, we stop our algorithm when the threshold reaches a value below which additional elments will not significantly improve the quality of the approximation, thereby ensuring that the total number of iterations of the algorithm is small.

Initially, the value of the threshold is set to $b = \max_{x \in X} f(\{x\})$. Due to the law of diminishing returns for submodular functions, $f_T(x) \leq b$ for all sets $T \subseteq X$ and elements $x \in X$. We begin with an empty set T_0 and at each stage choose a new element to add to the current set if its marginal value to the current set is greater than the threshold. When no element meets this criterion, we shrink the threshold by a factor of $(1 - \epsilon)$, stopping when it equals $\epsilon b/n$. Any elements whose marginal value is less than $\epsilon b/n$ would contribute a total of at most ϵb to the value of our solution, so we ignore them. Note that in some cases the threshold algorithm will terminate before it adds k elements to the set B.

We have the following proposition.

Proposition 2.2 (Badanidiyuru and Vondrák [1])**.** *Let $(T_i)_{i=0}^k$ be the sequence of subsets produced by the threshold algorithm, and let O be defined as in Proposition 2.1. Then*

$$f(T_{i+1}) - f(T_i) \geq \frac{(1-\epsilon)}{k}(f(O) - f(T_i)). \tag{2.3}$$

Note that Equation (2.3) immediately implies that the threshold algorithm yields a $(1 - 1/e - \epsilon)$-approximation by the same inductive argument used in the proof of Proposition 2.1. Indeed, Equation (2.3) is an exact analogue of Equation (2.2), and both equations can be proved in a similar fashion. The proof of Equation (2.2) relied on using the greedy choice to bound the marginal value of all other elements. In the case of the threshold algorithm, we cannot guarantee that our choice at each stage matches the choice the greedy algorithm would have made. However, because the threshold value changes only slightly from step to step, we can say that the marginal value of elements we add at each step is at most a $(1-\epsilon)$ factor away from the value of the greedy choice.

Proof of Proposition 2.2. As in the proof of Proposition 2.1, we order the elements of O arbitrarily and have by monotonicity that

$$\begin{aligned} f(O) \leq f(O \cup T_i) &= f(T_i) + \sum_{j=1}^{k} (f(T_i \cup \{o_1, \ldots, o_j\}) - f(T_i \cup \{o_1, \ldots, o_{j-1}\}) \\ &\leq f(T_i) + \sum_{j=1}^{k} f_{T_i}(o_j), \end{aligned}$$

where as in Proposition 2.1 the final inequality follows from submodularity. Note that if $o_j \in T_i$, then $f_{T_i}(o_j) = 0$, so in fact we have

$$f(O) \leq f(T_i) + \sum_{o \in O \setminus T_i} f_{T_i}(o). \tag{2.4}$$

Denote by y_{i+1} the element added by the threshold algorithm to the set T_i, and suppose that the current threshold at that time is t. If $o \notin T_i$, then in particular o did not meet the threshold at the previous stage of the algorithm, so its marginal value is at most $t/(1-\epsilon)$. On the other hand, the marginal value of y_{i+1} is at least t. We therefore have $f_{T_i}(o) \leq f_{T_i}(y_{i+1})/(1-\epsilon)$. Equation (2.4) then yields

$$f(O) \leq f(T_i) + (1-\epsilon)^{-1} \sum_{o \in O \setminus T_i} f_{T_i}(y_{i+1}) \leq f(T_i) + (1-\epsilon)^{-1} k f_{T_i}(y_{i+1}).$$

Since $f_{T_i}(y_{i+1}) = f(T_{i+1}) - f(T_i)$, the claim follows. □

The threshold algorithm therefore produces an approximation guarantee comparable with that of the greedy algorithm. Moreover, since the threshold algorithm only requires $O(\log_{1-\epsilon}(\epsilon/n)) = O(\frac{1}{\epsilon}\log(n/\epsilon))$ iterations, it has an

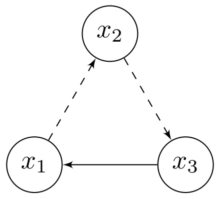

Figure 1: Dashed edges have weight ϵ, while the other edge has weight 1.

asymptotically better running time. In fact, the greedy algorithm and threshold algorithm often perform much better than their provable approximation bounds would suggest. In the conclusion, we briefly describe related work which seeks to quantify this puzzling phenomenon.

3 General submodular maximization

We now turn to the more difficult problem of maximizing a general submodular function. Like maximizing monotone submodular functions, maximizing a general submodular function is also NP-hard. Recall our example in Section 1.2, in which we reduced the NP-hard problem of finding a maximum cut to that of maximizing a submodular function. In fact, it has been shown by Lovász [6] that finding the exact maximum of a submodular function takes an exponential amount of time, even without the assumption that P $\neq$ NP.

A naïve greedy algorithm is not effective in the general case, since it is possible to construct examples where adding elements greedily performs arbitrarily poorly. Nevertheless, a type of greedy algorithm can find a constant-factor approximation in the general case. We present the approach of Buchbinder et al. [2], who derive a simple deterministic (1/3)-approximation algorithm, which is strengthened to a (1/2)-approximation using randomization. These algorithms are easy to implement and run in linear time.

Suppose throughout that we are working with a submodular function f defined on subsets of a ground set X. We will further assume that the elements of X are ordered: $x_1, x_2, \dots, x_n$.

We might suppose that we can greedily maximize f as follows: We maintain a set A, initially empty, that increases in size; for each element $x_i \in X$ in turn, we add x_i to A if doing so increases $f(A)$. Finally we output A. This algorithm does not yield a constant-factor approximation. For example, consider the problem Directed Max-Cut. Recall from Section 1.2 that the function that gives the value of a directed cut is submodular. The maximum directed cut of the graph in Figure 1 has value 1, but a greedy algorithm that considers the vertices in the order x_1, x_2, x_3 finds a cut of value ϵ. This greedy algorithm therefore performs arbitrarily poorly in the general case.

Another approach proceeds in the opposite direction: We maintain a set B, initially equal to X, that decreases in size; for each element $x_i \in X$ in turn,

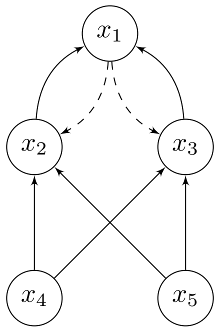

Figure 2: The dashed edges have weighted $1-\epsilon$ and the other edges have weight 1. In this graph, the deterministic algorithm in Buchbinder et al. [2] achieves a $1/3 + O(\epsilon)$ approximation. The algorithm finds a directed cut $\{x_2, x_3, x_4, x_5\}$ of value 2 whereas the maximum cut $\{x_1, x_4, x_5\}$ has value $6 - 2\epsilon$. Note that the vertices x_1, x_2, x_3, x_4, x_5 are considered by the algorithm in that order.

we remove x_i from B if doing so increases $f(B)$. Finally we output B. This "reverse greedy" algorithm also fails, and for similar reasons: if we consider the vertices of Figure 1 in the order x_3, x_2, x_1, we again produce a cut of weight ϵ.

Surprisingly, adopting both approaches at once yields an effective algorithm.

3.1 (1/3)-Approximation.

Our first algorithm proceeds deterministically as follows. We maintain both a set A and a set B, with A initialized to $\emptyset$ and B initialized to X. For each element $x_i \in X$ in turn, we either add x_i to A or remove x_i from B. Thus, after the ith step, A and B agree on the elements $x_1, x_2, \ldots, x_i$, and finally converge to the same set, which is our output. In order to determine whether to add x_i to A or remove it from B, we evaluate

$$a_i = f(A \cup \{x_i\}) - f(A)$$
$$b_i = f(B \backslash \{x_i\}) - f(B).$$

We add x_i to A if $a_i \geq b_i$ (the benefit from including x_i is greater than the benefit of taking it away), and otherwise we remove x_i from B.

Theorem 3.1 (Buchbinder et al. [2]). *This algorithm yields a* (1/3)*-approximation.*

Before outlining the proof of this theorem, we note that the example shown in Figure 2 shows the analysis is tight, where again we consider the special case of finding a maximum cut in a directed graph. In the case of the graph shown, the algorithm yields a cut $\{x_2, x_3, x_4, x_5\}$ of value 2 while the maximum value for a cut is $6 - 2\epsilon$, given by $\{x_1, x_4, x_5\}$.

Proof outline for Theorem 3.1. Let A_i be the state of set A after we have considered the vertices $x_1, \ldots, x_i$. Define B_i likewise. Thus, A_i equals A_{i-1} or $A_{i-1} \cup \{x_i\}$ and B_i equals B_{i-1} or $B_{i-1} \backslash \{x_i\}$. Let $O \subseteq X$ be a vertex set for which the submodular function $f(O)$ is maximized. Let $O_i = (O \cup A_i) \cap B_i$; hence O_i agrees with A_i and B_i on the elements $x_1, \ldots, x_i$ and agrees with O on $x_{i+1}, \ldots, x_n$.

Lemma 3.2. *For each i, we have*

$$f(O_{i-1}) - f(O_i) \leq (f(A_i) - f(A_{i-1})) + (f(B_i) - f(B_{i-1})) . \tag{3.1}$$

The proof of this lemma proceeds routinely by manipulating the definition of submodularity and considering the procedure by which A_i and B_i are updated. Once the lemma is proven, the theorem follows by summing Equation 3.1 from $i = 0$ to n; the summands "telescope" to give:

$$f(O_0) - f(O_n) \leq (f(A_n) - f(A_0)) + (f(B_n) - f(B_0)) .$$

Note that

$$A_0 = \emptyset, \quad B_0 = X, \quad O_0 = O, \quad A_n = B_n = O_n.$$

Hence, our inequality simplifies to

$$f(O) \leq 3f(A_n),$$

which proves that the algorithm gives a (1/3)-approximation ratio. □

3.2 (1/2)-Approximation.

The above algorithm can be strengthened by including the following element of randomness. At the ith step, we decide randomly whether to include x_i in A or remove it from B. If $a_i, b_i > 0$, then we add x_i to A with probability $a_i/(a_i + b_i)$, and remove it from B with probability $b_i/(a_i + b_i)$. If $a_i > 0$ and $b_i \leq 0$, then we add x_i to A; likewise if $b_i > 0$ and $a_i \leq 0$, then we remove x_i from B. Otherwise, then we decide to include x_i in A; the same result is obtained if we decide to remove x_i from B.

Theorem 3.3 (Buchbinder et al. [2])**.** *This algorithm yields a (1/2)-approximation.*

Proof outline. Define A_i, B_i, O, O_i as in the deterministic algorithm. The key observation in the proof is that Lemma 3.2 can be replaced by a stronger version:

Lemma 3.4. *For each i, we have*

$$\begin{aligned} &2 \cdot \mathbb{E}[f(O_{i-1}) - f(O_i)] \\ &\leq \mathbb{E}\left[f(A_i) - f(A_{i-1})\right] + \mathbb{E}\left[f(B_i) - f(B_{i-1})\right] . \end{aligned} \tag{3.2}$$

From this lemma, the proof of Theorem 3.3 proceeds by a telecoping sum, as in the proof of Theorem 3.1. To prove the lemma, we divide into cases based on the signs of a_i and b_i. For simplicity, we present only the case of $a_i, b_i \geq 0$, which is also the most interesting.

Observe that

$$\begin{aligned}
&\mathbb{E}\left[f(A_i) - f(A_{i-1})\right] + \mathbb{E}\left[f(B_i) - f(B_{i-1})\right] \\
&= \frac{a_i}{a_i+b_i}\left[f(A_{i-1} \cup \{x_i\}) - f(A_{i-1})\right] + \frac{b_i}{a_i+b_i}\left[f(B_{i-1} \backslash \{x_i\}) - f(B_{i-1})\right] \\
&= \frac{a_i^2+b_i^2}{a_i+b_i}.
\end{aligned}$$

On the other side of Equation (3.2), we can use submodularity to show that

$$\begin{aligned}
&2 \cdot \mathbb{E}[f(O_{i-1}) - f(O_i)] \\
&= 2\frac{a_i}{a_i+b_i}\mathbb{E}\left[f(O_{i-1}) - f(O_{i-1} \cup \{x_i\})\right] \\
&+ 2\frac{b_i}{a_i+b_i}\mathbb{E}\left[f(O_{i-1}) - f(O_{i-1} \backslash \{x_i\})\right] \\
&\leq \frac{2a_ib_i}{a_i+b_i}.
\end{aligned}$$

The intuition for this expression is that either

$$\begin{aligned}
f(O_{i-1}) - f(O_{i-1} \cup \{x_i\}) &= 0 \qquad \text{and} \\
f(O_{i-1}) - f(O_{i-1} \backslash \{x_i\}) &= a_i,
\end{aligned}$$

or else

$$\begin{aligned}
f(O_{i-1}) - f(O_{i-1} \cup \{x_i\}) &= b_i \qquad \text{and} \\
f(O_{i-1}) - f(O_{i-1} \backslash \{x_i\}) &= 0.
\end{aligned}$$

Equation 3.2 now follows from since

$$\frac{2a_ib_i}{a_i+b_i} \leq \frac{a_i^2+b_i^2}{a_i+b_i}.$$

(This inequality reduces to $(a_i - b_i)^2 \geq 0$.) This completes our proof of the lemma, and Theorem 3.3 follows. □

3.3 Improvements to the algorithm

This (1/2)-approximation cannot be strengthened in the general case, as a result by Feige et al. [4] proves that any $(1/2 + \epsilon)$-approximation must require exponentially many queries to the submodular function. However, stronger results may be possible for special problems of submodular optimization. Buchbinder et al. present a (3/4)-approximation algorithm for the problems of Submodular

Max-SAT and Submodular Welfare, which we do not define here. One of the ingredients in this improvement is a randomization of the order in which elements of X are considered.

We have considered improvements to the algorithm for the problem of Directed Max-Cut. Figure 2 shows that the analysis is tight for (1/3)-approximation, but this example relies on the vertices having been given in a particular order. If we randomize the order of the elements of X, we believe that much stronger results can be achieved in practice. Pathological orderings like the one in Figure 2 seem to be quite unlikely. Indeed, our data indicates that for arbitrary graphs, applying the algorithm in Section 3 to a graph whose vertices are given in a random order yields solutions which are very nearly optimal. Once again, we note the practical efficacy of greedy approaches above and beyond their provable approximation guarantees.

4 Conclusion

Maximizing submodular functions is a problem with a great deal of theoretical and practical significance. Most modern approaches are "greedy" to some extent, in that they recover global approximation guarantees from optimal local choices. The algorithms we have considered in this paper improve upon the naïve approach by carefully specifying how these optimal local choices are made.

In this paper, we have focused on purely combinatorial algorithms, but another strain of recent work, launched by Calinescu et al. in [3], focuses on continuous maximization algorithms. These algorithms are also essentially greedy. In fact, an algorithm in [1] is controlled by a parameter δ, which smoothly interpolates between the combinatorial greedy algorithm when $\delta = 1$ and a fully continuous greedy algorithm when $\delta = 0$. For all its simplicity, the greedy approach still forms the foundation of modern techniques.

Though many of the bounds we have presented are provably tight, it is worth noting that the algorithms we have discussed often perform substantially better in practice. For instance, as mentioned in Section 3, randomization alone can improve performance significantly on realistic data, and refined algorithms can improve the approximation ratio significantly in special cases such as Submodular Max-SAT and Submodular Welfare. Recent work (see, for instance, [10]) attempts to quantify such phenomena by adopting additional assumptions about the submodular functions in question, such as that they are symmetric or of bounded curvature. This seems a promising avenue for future work.

References

[1] A. Badanidiyuru and J. Vondrák, *Fast algorithms for maximizing submodular functions*, SODA (2014).

[2] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz, *A tight linear time (1/2)-approximation for unconstrained submodular maximization*, 2012 IEEE 53rd Annu. Symp. Found. Comput. Sci. (October 2012), 649–658.

[3] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, *Maximizing a monotone submodular function subject to a matroid constraint*, SIAM J. Comput. (2011), 1–28.

[4] U. Feige, V.S. Mirrokni, and J. Vondrák, *Maximizing non-monotone submodular functions*, SIAM J. Comput. (2011).

[5] M. Grötschel, L. Lovász, and A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica **1** (1981), no. 2, 169–197.

[6] L. Lovász, *The matroid matching problem*, Algebraic methods in graph theory, 1981, pp. 495–517.

[7] ______, *Submodular functions and convexity*, Math. program. state art, 1983, pp. 235–257.

[8] G.L. Nemhauser and L.A. Wolsey, *Best algorithms for approximating the maximum of a submodular set function*, Mathematics of operations research **3** (1978), no. 3, 177–188.

[9] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher, *An analysis of approximations for maximizing submodular set functionsi*, Mathematical Programming **14** (1978), no. 1, 265–294.

[10] M. Sviridenko, J. Vondrák, and J. Ward, *Optimal approximation for submodular and supermodular optimization with bounded curvature* (2014), 1–15.