

# Submodular Approximation: Sampling-Based Algorithms and Lower Bounds\*

Zoya Svitkina

Lisa Fleischer

Department of Computer Science, Dartmouth College  
{zoya, lkf}@cs.dartmouth.edu

## Abstract

We introduce several generalizations of classical computer science problems obtained by replacing simpler objective functions with general submodular functions. The new problems include submodular load balancing, which generalizes load balancing or minimum-makespan scheduling, submodular sparsest cut and submodular balanced cut, which generalize their respective graph cut problems, as well as submodular function minimization with a cardinality lower bound. We establish upper and lower bounds for the approximability of these problems with a polynomial number of queries to a function-value oracle. The approximation guarantees for most of our algorithms are of the order of  $\sqrt{n}/\ln n$ . We show that this is the inherent difficulty of the problems by proving matching lower bounds.

We also give an improved lower bound for the problem of approximately learning a monotone submodular function. In addition, we present an algorithm for approximately learning submodular functions with special structure, whose guarantee is close to the lower bound. Although quite restrictive, the class of functions with this structure includes the ones that are used for lower bounds both by us and in previous work. This demonstrates that if there are significantly stronger lower bounds for this problem, they rely on more general submodular functions.

## 1 Introduction

A function  $f$  defined on subsets of a ground set  $V$  is called *submodular* if for all subsets  $S, T \subseteq V$ ,  $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$ . Submodularity is a discrete analog of convexity. It also shares some nice properties with concave functions, as it captures decreasing marginal returns. Submodular functions generalize cut functions of graphs and rank functions of matrices and matroids, and arise in a variety of applications including facility location, assignment, scheduling, and network design.

In this paper, we introduce and study several generalizations of classical computer science problems. These new problems have a general submodular function in their objectives, in place of much simpler functions in the objectives of their classical counterparts. The problems include *submodular load balancing*, which generalizes load balancing or minimum-makespan scheduling, and *submodular minimization with cardinality lower bound*, which generalizes the minimum knapsack problem. In these two problems, the size of a collection of items, instead of being just a sum of their individual sizes, is now a submodular function. Two other new problems are *submodular sparsest cut* and *submodular balanced cut*, which generalize their respective graph cut problems. Here, a general submodular function replaces the graph cut function, which itself is a well-known special case of a submodular function. The last problem that we study is *learning a submodular function*, which was recently introduced by Goemans, Harvey, Kleinberg, and Mirrokni [8, 11]. All of these problems are defined on a set  $V$  of  $n$  elements with a nonnegative submodular function  $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ . Since the amount of information necessary to convey a general submodular function may be exponential in  $n$ , we rely on value-oracle access to  $f$  to develop algorithms with running time polynomial in  $n$ . A *value oracle* for  $f$  is a black box that, given a subset  $S$ , returns the value  $f(S)$ . The following are formal definitions of the problems.

**Submodular Sparsest Cut (SSC):** Given a set of unordered pairs  $\{\{u_i, v_i\} \mid u_i, v_i \in V\}$ , each with a demand  $d_i > 0$ , find a subset  $S \subseteq V$  minimizing  $f(S) / \sum_{i: |S \cap \{u_i, v_i\}|=1} d_i$ . The denominator is the amount of demand separated by the “cut”  $(S, \bar{S})$ <sup>1</sup>. We also consider *uniform SSC*, in which all pairs of nodes have demand equal to one. In this case the objective function is to minimize  $f(S) / |S| |\bar{S}|$ . We focus on a closely related formulation of the problem, known as the minimum-quotient cut in the case of graphs [19], which minimizes  $f(S) / \min(|S|, |\bar{S}|)$ . The objective functions of the two problems are always within a factor of at least  $n/2$  and at most  $n$  from each other, so a  $\gamma$ -approximation for one provides a  $2\gamma$ -approximation

\*This work supported in part by NSF grant CCF-0728869.

<sup>1</sup>For any set  $S \subseteq V$ , we use  $\bar{S}$  to denote its complement set,  $V \setminus S$ .

for the other. A version intermediate between uniform SSC and general SSC is *weighted* SSC, in which each  $v \in V$  has a non-negative weight  $w(v)$ , and the demand between any pair of elements  $(u, v)$  is equal to the product  $w(u) \cdot w(v)$ .

**Submodular  $b$ -Balanced Cut (SBC):** Given a weight function  $w : V \rightarrow \mathbb{R}_{\geq 0}$ , a cut  $(S, \bar{S})$  is called  $b$ -balanced (for  $b \leq \frac{1}{2}$ ) if  $w(S) \geq b \cdot w(V)$  and  $w(\bar{S}) \geq b \cdot w(V)$ , where  $w(S) = \sum_{v \in S} w(v)$ . The goal of the problem is to find a  $b$ -balanced cut  $(S, \bar{S})$  that minimizes  $f(S)$ .

**Submodular Minimization with Cardinality Lower Bound (SML):** For a given  $W \geq 0$ , find a subset  $S \subseteq V$  with  $|S| \geq W$  that minimizes  $f(S)$ . A generalization with 0-1 weights  $w : V \rightarrow \{0, 1\}$  is to find  $S$  with  $w(S) \geq W$  minimizing  $f(S)$ .

**Submodular Load Balancing (SLB):** The *uniform* version is to find, given a monotone<sup>2</sup> submodular function  $f$  and a positive integer  $m$ , a partition of  $V$  into  $m$  sets,  $V_1, \dots, V_m$  (some possibly empty), so as to minimize  $\max_i f(V_i)$ . The *non-uniform* version is to find, for  $m$  monotone submodular functions  $f_1, \dots, f_m$  on  $V$ , a partition  $V_1, \dots, V_m$  that minimizes  $\max_i f_i(V_i)$ .

**Learning a Submodular Function:** Produce a function  $\hat{f}$  (not necessarily submodular) that for all  $S \subseteq V$  satisfies  $\hat{f}(S) \leq f(S) \leq \gamma(n)\hat{f}(S)$ , with approximation ratio  $\gamma(n) \geq 1$  as small as possible. We also consider the special case of monotone two-partition functions, which we define as follows. A submodular function  $f$  is a *two-partition* (2P) function if there is a set  $R \subseteq V$  such that for all sets  $S$ , the value of  $f(S)$  depends only on the sizes  $|S \cap R|$  and  $|S \cap \bar{R}|$ .

## 1.1 Motivation

Submodular functions arise in a variety of contexts, often in optimization settings. The problems that we define in this paper use submodular functions to generalize some of the best-studied problems in computer science. These generalizations capture many variants of their corresponding classical problems. For example, the submodular sparsest and balanced cut problems generalize not only graph cuts, but also hypergraph cuts. In addition, they may be useful as subroutines for solving other problems, in the same way that sparsest and balanced cuts are used for approximating graph problems, such as the minimum cut linear arrangement, often as part of divide-and-conquer schemes. The SML problem can model a scenario in which costs follow economies of scale, and a certain number of items has to be bought at the minimum total cost. An example application of SLB is compressing and storing files on multiple hard drives or servers in a load-balanced way. Here the size of a compressed collection of files may be much smaller than the sum of individual file sizes, and modeling it by a monotone

<sup>2</sup>A function  $f$  is monotone if  $f(S) \leq f(T)$  whenever  $S \subseteq T$ .

submodular function is reasonable considering that entropy function is known to be monotone and submodular [6].

## 1.2 Related work

Because of the relation of submodularity to cut functions and matroid rank functions, and their exhibition of decreasing marginal returns, there has been substantial interest in optimization problems involving submodular functions. Finding the set that has the minimum function value is a well-studied problem that was first shown to be polynomially solvable using the ellipsoid method [9, 10]. Further research has yielded several more combinatorial approaches [16, 17, 18, 23, 24, 26].

Submodular functions arise in facility location and assignment problems, and this has spawned interest in the problem of finding the set with the maximum function value. Since this is NP-hard, research has focused on approximation algorithms for maximizing submodular functions, perhaps subject to a cardinality constraint or other simple constraint [3, 5, 22, 27]. There has been research on other optimization problems that use submodular functions or their minimization, including [13, 15, 30, 31, 32]. Zhao et al. [33] study a submodular multiway partition problem, which is similar to our SLB problem, except that the objective function is the sum of the functions of subsets, as opposed to the maximum.

Since it is impossible to learn a general submodular function exactly without looking at the function value on all (exponentially many) subsets [4], there has been recent interest in approximating submodular functions with a polynomial number of value oracle queries. Some lower bounds on the approximation guarantees achievable in this model are given in [8, 11].

All of the optimization problems that we consider in this paper are known to be NP-hard even when the objective function can be expressed compactly as a linear or graph-cut function. While there is an FPTAS for the minimum knapsack problem [7], the best approximation for load balancing on uniform machines is a PTAS [14], and on unrelated machines the best possible upper and lower bounds are constants [20]. The best approximation known for the sparsest cut problem is  $O(\sqrt{\log n})$  [1, 2], and the balanced cut problem is approximable to a factor of  $O(\log n)$  [25]. For the special case of SML on graphs, introduced in [29], an  $O(\log n)$  approximation is possible using the recent results of Räcke [25].

## 1.3 Our results and techniques

We establish upper and lower bounds for the approximability of the problems listed above. Surprisingly, these factors are quite high. Whereas the corresponding classical

problems are approximable to constant or logarithmic factors, the guarantees that we prove for most of our algorithms are of the order of  $\sqrt{\frac{n}{\ln n}}$ . We show that this is the inherent difficulty of these problems by proving matching (or, in some cases, almost matching) lower bounds. Our lower bounds are unconditional, and rely on the difficulty of distinguishing different submodular functions by performing only a polynomial number of queries in the oracle model. The proofs are based on the techniques in [5, 8]. To prove the upper bounds, we present randomized approximation algorithms which use their randomness for sampling subsets of the ground set of elements. We show that with relatively high probability (inverse polynomial), a sample can be obtained such that its overlap with the optimal set is significantly higher than expected. Using the samples, the algorithms employ submodular function minimization to find candidate solutions. This is done in such a way that if the sample does indeed have a large overlap with the optimal set, then the solution satisfies the algorithm’s guarantee.

For SSC and uniform SLB, we show that they can be approximated to a  $\Theta\left(\sqrt{\frac{n}{\ln n}}\right)$  factor. For SBC, we use the weighted SSC as a subroutine, which allows us to obtain a bicriteria approximation in a similar way as Leighton and Rao [19] do for graphs. For SML, we consider bicriteria results. For  $\rho \geq 1$  and  $0 < \sigma \leq 1$ , a  $(\rho, \sigma)$ -approximation is an algorithm that outputs a set  $S$  such that  $f(S) \leq \rho B$  and  $w(S) \geq \sigma W$ , whenever the input instance contains a set  $U$  with  $f(U) \leq B$  and  $w(U) \geq W$ . We present a lower bound showing that there is no  $(\rho, \sigma)$  approximation for any  $\rho$  and  $\sigma$  with  $\frac{\rho}{\sigma} = o\left(\sqrt{\frac{n}{\ln n}}\right)$ . For 0-1 weights, we obtain a  $(5\sqrt{\frac{n}{\ln n}}, \frac{1}{2})$  approximation. This algorithm can be used to obtain an  $O(\sqrt{n \ln n})$  approximation for non-uniform SLB.

We briefly note here that one can consider the problem of minimizing a submodular function with an *upper* bound on cardinality (i.e., minimize  $f(S)$  subject to  $|S| \leq W$ ). For this problem, we do not know of any lower bounds other than NP-hardness, and a  $(\frac{1}{\alpha}, \frac{1}{1-\alpha})$  bicriteria approximation is possible for any  $0 < \alpha < 1$ , using techniques in [12].

For learning *monotone* submodular functions, our lower bound is  $\Omega\left(\sqrt{\frac{n}{\ln n}}\right)$ , which improves the bound of  $\Omega\left(\frac{\sqrt{n}}{\ln n}\right)$  in [8], and matches the lower bound for learning arbitrary submodular functions, also in [8]. Our lower bound proof for learning, as well as those in [8], use 2P functions, and thus still hold for this special case. We show that learning monotone 2P functions can be approximated within a factor  $O(\sqrt{n})$ . Besides leaving a relatively small gap between the upper and lower bounds, this shows that if much stronger lower bounds for the learning problem exist, they rely on more general submodular functions.

For the problems studied in this paper, our lower bounds show the impossibility of constant or even polylogarithmic approximations in the value oracle model. This means that

in order to obtain better results for specific applications, one has to resort to more restricted models, avoiding the full generality of arbitrary submodular functions.

## 2 Preliminaries

In the analysis of our algorithms, we repeatedly use the facts that the sum of submodular functions is submodular, and that submodular functions can be minimized in polynomial time. For example, this allows us to minimize (over  $T \subseteq V$ ) expressions like  $f(T) - \alpha \cdot |T \cap S|$ , where  $\alpha$  is a constant and  $S$  is a fixed subset of  $V$ . Some proofs are omitted from this extended abstract and can be found in [28].

We present our algorithms by providing a *randomized relaxed decision procedure* for each of the problems. Given an instance of a minimization problem, a target value  $B$ , and a probability  $p$ , this procedure either declares that the problem is infeasible (outputs *fail*), or finds a solution to the instance with objective value at most  $\gamma B$ , where  $\gamma$  is the approximation factor. We say that an instance is feasible if it has a solution with cost strictly less than  $B$  (we use strict inequality for technical reasons; this can be avoided by adding a small value  $\varepsilon > 0$  to  $B$ ). The guarantee provided with each decision procedure is that for any feasible instance, it outputs a  $\gamma$ -approximate solution with probability at least  $p$ . On an infeasible instance, either of the two outcomes is allowed. Randomized relaxed decision procedures can be turned into randomized approximation algorithms by finding upper and lower bounds for the optimum and performing binary search. Our algorithms run in time polynomial in  $n$  and  $\ln \frac{1}{1-p}$ .

Let us say that an algorithm *distinguishes* two functions  $f_1$  and  $f_2$  if it produces different output if given (an oracle for)  $f_1$  as input than if given (an oracle for)  $f_2$ . The following result is used for obtaining all of our lower bounds.

**Lemma 2.1** *Let  $f_1$  and  $f_2$  be two set functions, with  $f_2$ , but not  $f_1$ , parametrized by a string of random bits  $r$ . If for any set  $S$ , chosen without knowledge of  $r$ , the probability (over  $r$ ) that  $f_1(S) \neq f_2(S)$  is  $n^{-\omega(1)}$ , then any algorithm that makes a polynomial number of oracle queries has probability at most  $n^{-\omega(1)}$  of distinguishing  $f_1$  and  $f_2$ .*

**Proof.** We use reasoning similar to [5]. Consider first a deterministic algorithm and the computation path that it follows if it receives the values of  $f_1$  as answers to all its oracle queries. Note that this is a single computation path that does not depend on  $r$ , because  $f_1$  does not depend on  $r$ . On this path the algorithm makes some polynomial number of oracle queries, say  $n^a$ . Using the union bound, we know that the probability that  $f_1$  and  $f_2$  differ on any of these  $n^a$  sets is at most  $n^a \cdot n^{-\omega(1)} = n^{-\omega(1)}$ . So, with probability at least  $1 - n^{-\omega(1)}$ , if given either  $f_1$  or  $f_2$  as input, the algorithm

only queries sets for which  $f_1 = f_2$ , and therefore stays on the same computation path, producing the same answer in both cases.

A randomized algorithm can be viewed as a distribution over a set of deterministic algorithms. Since, by the discussion above, each of these deterministic algorithms has probability at most  $n^{-\omega(1)}$  of distinguishing  $f_1$  and  $f_2$ , the randomized algorithm as a whole also has probability at most  $n^{-\omega(1)}$  of distinguishing these two functions.  $\square$

### 3 Submodular sparsest and balanced cuts

#### 3.1 Lower bounds

To show a lower bound for SSC and SBC, we use the following result from [8] and Lemma 2.1.

**Lemma 3.1 (Goemans et al. [8])** *Fix an arbitrary subset  $S \subseteq V$ , and then let  $R$  be a random subset of  $V$  of size  $n/2$ . Then for any  $\varepsilon$  such that  $\varepsilon^2 = \frac{1}{n} \cdot \omega(\ln n)$ , for  $\beta = \frac{n}{4}(1 + \varepsilon)$ , and for the submodular<sup>3</sup> functions*

$$\begin{aligned} f_1(S) &= \min\left(|S|, \frac{n}{2}\right) - \frac{|S|}{2} \\ f_2(S) &= \min\left(\beta + |S \cap \bar{R}|, |S|, \frac{n}{2}\right) - \frac{|S|}{2}, \end{aligned}$$

the probability (over  $R$ ) that  $f_1(S) \neq f_2(S)$  is  $n^{-\omega(1)}$ .

**Corollary 3.2** *Any algorithm that makes a polynomial number of oracle queries has probability at most  $n^{-\omega(1)}$  of distinguishing the functions  $f_1$  and  $f_2$  in Lemma 3.1.*

We now establish the hardness of SSC and SBC problems. For concreteness, assume that the output of an approximation algorithm for one of these problems consists of a set  $S \subseteq V$  as well as its objective function value.

**Theorem 3.3** *The uniform SSC and the SBC problems cannot be approximated to a ratio  $o\left(\sqrt{\frac{n}{\ln n}}\right)$  in the oracle model with polynomial number of queries.*

**Proof.** Suppose for the sake of contradiction that there is a polynomial-time  $\gamma$ -approximation algorithm for the uniform SSC problem, for some  $\gamma = o\left(\sqrt{\frac{n}{\ln n}}\right)$ , that succeeds with high probability. Let us set  $\varepsilon = \frac{1}{2\gamma}$ , which satisfies  $\varepsilon^2 = \frac{1}{n} \cdot \omega(\ln n)$ . If this algorithm is given the function  $f_2$  of Lemma 3.1 as input, then with high probability it has to

<sup>3</sup>To see that these functions are submodular, it is helpful to consider an equivalent definition of submodularity: for all  $a, b \in V$  and  $S \subset V$ ,  $f(S \cup \{a\}) - f(S) \leq f(S \cup \{a\} - \{b\}) - f(S - \{b\})$ . Then it is straightforward to verify for these, and submodular functions appearing later in the paper, that if adding  $a$  to  $S - \{b\}$  increases the function value by  $x$ , adding  $a$  to  $S$  increases the function value by at most  $x$ .

output a set  $S$  with ratio  $\frac{f_2(S)}{\min(|S|, |S^c|)} \leq \frac{\gamma\varepsilon}{2} = \frac{1}{4}$ , since the optimal uniform sparsest cut for  $f_2$ , achieved by the set  $R$ , has ratio  $\frac{\beta - n/4}{n/2} = \frac{\varepsilon}{2}$ . However, for the function  $f_1$ , all sets have ratio equal to  $\frac{f_1(S)}{\min(|S|, |S^c|)} = \frac{1}{2}$ . So if the algorithm is given  $f_1$  as input, its output value is  $\frac{1}{2}$ , thus differing from the case of  $f_2$ . But this contradicts Corollary 3.2.

A similar argument establishes the SBC lower bound.  $\square$

#### 3.2 Algorithm for uniform SSC

Uniform SSC is a special case of general SSC, so our algorithm in Section 3.3 applies to the uniform case as well. We present a separate algorithm for uniform SSC first (Algorithm 1), because it is simpler and has a slightly better guarantee. The algorithm works under the assumption that the size of the optimal set  $U^*$  is at most  $n/2$ , which we use in the analysis. To handle the possible case that  $|U^*| > n/2$ , we run the algorithm twice, once for  $f$ , and once for  $\bar{f}$ , which is defined as  $\bar{f}(S) = f(\bar{S})$ , and is submodular whenever  $f$  is. The output is either the set produced by the first run of the algorithm, or the complement of the set found by the second run, whichever has a better ratio.

---

**Algorithm 1** Uniform SSC. Input:  $V, f, B, p$

---

- 1: Let  $\alpha = B \cdot \sqrt{\frac{n}{\ln n}}$
  - 2: **for**  $\frac{n^2}{\varepsilon^2} \ln\left(\frac{1}{1-p}\right)$  iterations **do**
  - 3:     Choose a uniformly random  $S \subset V$  of size  $n/2$
  - 4:     Let  $T \subseteq V$  minimize  $f(T) - \alpha|T \cap S| + \alpha|T \cap \bar{S}|$
  - 5:     **if**  $f(T) - \alpha|T \cap S| + \alpha|T \cap \bar{S}| < 0$ , **return**  $T$
  - 6: **end for**
  - 7: **return fail**
- 

The idea of the algorithm is to sample a random set  $S$ , hoping that the overlap with the optimal set is high, and the overlap with the complement of the optimal set is low. Then the minimization on line 4 “encourages” the inclusion of elements from  $S$  and “discourages” the elements from  $\bar{S}$ . If the sample  $S$  is successful, then this optimization finds an approximate solution.

**Theorem 3.4** *For any feasible instance of the uniform SSC problem, Algorithm 1 returns a solution of cost at most  $\sqrt{\frac{n}{\ln n}} \cdot B$  with probability at least  $p$ .*

The proof of this theorem follows from several lemmas.

**Lemma 3.5** *If for some  $T \subseteq V$  and a set  $S$  with  $|S| = n/2$  it holds that  $f(T) - \alpha \cdot |T \cap S| + \alpha \cdot |T \cap \bar{S}| < 0$ , then  $\frac{f(T)}{\min(|T|, |\bar{T}|)} < \alpha$ .*

**Proof.** We claim that  $\min(|T|, |\bar{T}|) \geq |T \cap S| - |T \cap \bar{S}|$ , which follows because  $|S| = |\bar{S}| = n/2$ . Then, from the

assumption of the lemma,  $f(T) - \alpha \min(|T|, |\bar{T}|) \leq f(T) - \alpha (|T \cap S| - |T \cap \bar{S}|) < 0$ . Since  $f$  is non-negative, we must have  $\min(|T|, |\bar{T}|) > 0$ . This allows us to rearrange the terms, obtaining  $\frac{f(T)}{\min(|T|, |\bar{T}|)} < \alpha$ .  $\square$

**Lemma 3.6** *Let  $U$  be any subset of  $V$  with  $|U| \leq n/2$ , let  $m = |U|$ , and suppose that  $S$  is a random subset of  $V$  of size  $n/2$ . Then with probability<sup>4</sup> at least  $\frac{c^2}{n^2}$ ,  $|U \cap S| \geq \frac{m}{2} \left(1 + \sqrt{\frac{\ln n}{n}}\right)$ .*

**Proof of Theorem 3.4 :** Suppose that the given instance of the uniform SSC problem is feasible, and let  $U^*$  be a set such that  $f(U^*)/|U^*| < B$ . Let  $m = |U^*|$ . As mentioned earlier, we assume that  $m \leq n/2$ .

Consider one iteration of the loop in Algorithm 1, and let  $S$  be the random set chosen in this iteration. By Lemma 3.6, with probability at least  $\frac{c^2}{n^2}$ ,  $|U^* \cap S| \geq m \left(\frac{1}{2} + \frac{1}{2} \sqrt{\frac{\ln n}{n}}\right)$ .

If this is the case, then  $|U^* \cap \bar{S}| \leq m \left(\frac{1}{2} - \frac{1}{2} \sqrt{\frac{\ln n}{n}}\right)$ , and  $f(U^*) - \alpha \cdot |U^* \cap S| + \alpha \cdot |U^* \cap \bar{S}|$  is at most

$$f(U^*) - \alpha \cdot m \sqrt{\frac{\ln n}{n}} \leq f(U^*) - B \sqrt{\frac{n}{\ln n}} \cdot m \sqrt{\frac{\ln n}{n}} < 0.$$

So the set  $T$ , chosen by Algorithm 1 to minimize expression on line 4, will satisfy  $f(T) - \alpha \cdot |T \cap S| + \alpha \cdot |T \cap \bar{S}| < 0$ , and, by Lemma 3.5,  $\frac{f(T)}{\min(|T|, |\bar{T}|)} < \alpha = B \sqrt{\frac{n}{\ln n}}$ . When the loop is repeated  $\frac{n^2}{c^2} \ln\left(\frac{1}{1-p}\right)$  times, the probability that the algorithm finds a solution is at least  $p$ .  $\square$

### 3.3 Algorithm for general SSC

Our approach for solving this version is similar to the one we used for the uniform case, but now we use a random set  $S$  to assign weights to nodes (see Algorithm 2). For each demand pair separated by the set  $S$ , we add a positive weight of  $d_i$  to its node that is in  $S$ , and a negative weight of  $-d_i$  to its node that is outside of  $S$ . This biases the subsequent function minimization to separate the demand pairs that are on different sides of  $S$ .

The following lemma is analogous to Lemma 3.5.

**Lemma 3.7** *If for some set  $T \subseteq V$ , it holds that  $f(T) - \alpha \cdot \sum_{v \in T} w(v) < 0$ , then*

$$\frac{f(T)}{\sum_{i:|T \cap \{u_i, v_i\}|=1} d_i} < \alpha.$$

<sup>4</sup>We use a constant  $c = (e^2 \sqrt{2\pi})^{-1}$  throughout the paper.

---

**Algorithm 2** Submodular sparsest cut. Input:  $V, f, d, B, p$

---

```

1: for  $\frac{4n^2}{c} \ln\left(\frac{1}{1-p}\right)$  iterations do
2:   Choose a uniformly random set  $S \subseteq V$ 
3:   for each  $v \in V$ , initialize a weight  $w(v) = 0$ 
4:   for each pair  $\{u_i, v_i\}$  with  $|\{u_i, v_i\} \cap S| = 1$  do
5:     Set  $s_i = \{u_i, v_i\} \cap S$ ;  $w(s_i) \leftarrow w(s_i) + d_i$ 
6:     Set  $t_i = \{u_i, v_i\} \setminus S$ ;  $w(t_i) \leftarrow w(t_i) - d_i$ 
7:   end for
8:   Let  $\alpha = 4\sqrt{\frac{n}{\ln n}} \cdot B$ 
9:   Let  $T \subseteq V$  minimize  $f(T) - \alpha \cdot \sum_{v \in T} w(v)$ 
10:  if  $f(T) - \alpha \cdot \sum_{v \in T} w(v) < 0$ , return  $T$ 
11: end for
12: return fail

```

---

**Proof.** We have  $\sum_{v \in T} w(v) =$

$$\begin{aligned} &= \sum_{i:s_i \in T} d_i - \sum_{i:t_i \in T} d_i = \sum_{i:s_i \in T, t_i \notin T} d_i - \sum_{i:t_i \in T, s_i \notin T} d_i \\ &\leq \sum_{i:s_i \in T, t_i \notin T} d_i \leq \sum_{i:|T \cap \{u_i, v_i\}|=1} d_i \end{aligned}$$

Now using the assumption of the lemma we have

$$f(T) - \alpha \sum_{i:|T \cap \{u_i, v_i\}|=1} d_i \leq f(T) - \alpha \sum_{v \in T} w(v) < 0.$$

Function  $f$  is non-negative, so  $\sum_{i:|T \cap \{u_i, v_i\}|=1} d_i > 0$ . Rearranging,  $f(T)/\sum_{i:|T \cap \{u_i, v_i\}|=1} d_i < \alpha$ .  $\square$

Assuming that the input instance is feasible, let  $U^*$  be a set with size  $m = |U^*|$ , separated demand  $D^* = \sum_{i:|U^* \cap \{u_i, v_i\}|=1} d_i$ , and value  $f(U^*)/D^* < B$ .

**Lemma 3.8** *In any one iteration of the outer loop of Algorithm 2, the probability that  $\sum_{v \in U^*} w(v) \geq D^* \cdot \frac{1}{4} \sqrt{\frac{\ln n}{n}}$  is at least  $\frac{c}{4n^2}$ .*

**Theorem 3.9** *For any feasible SSC instance, Algorithm 2 returns a solution with cost at most  $4\sqrt{\frac{n}{\ln n}} \cdot B$ , with probability at least  $p$ .*

**Proof.** By Lemma 3.8, the inequality  $\sum_{v \in U^*} w(v) \geq D^* \cdot \frac{1}{4} \sqrt{\frac{\ln n}{n}}$  holds with probability at least  $c/4n^2$  in each iteration. Then the probability that it holds in any of the  $\frac{4n^2}{c} \ln\left(\frac{1}{1-p}\right)$  iterations is at least  $p$ . Now, assuming that it does hold, the algorithm finds a set  $T$  such that

$$\begin{aligned} f(T) - \alpha \cdot \sum_{v \in T} w(v) &\leq f(U^*) - \alpha \cdot \sum_{v \in U^*} w(v) \\ &\leq f(U^*) - \left(4\sqrt{\frac{n}{\ln n}} \cdot B\right) \left(D^* \cdot \frac{1}{4} \sqrt{\frac{\ln n}{n}}\right) < 0. \end{aligned}$$

By Lemma 3.7,  $f(T)/\sum_{i:|T \cap \{u_i, v_i\}|=1} d_i < \alpha = B \cdot 4\sqrt{\frac{n}{\ln n}}$ , so  $T$  is the required approximate solution.  $\square$

### 3.4 Submodular balanced cut

For submodular balanced cut, we use as a subroutine the weighted SSC problem that can be approximated to a factor  $\gamma = O\left(\sqrt{\frac{n}{\ln n}}\right)$  using Algorithm 2. This allows us to obtain a bicriteria approximation for SBC in a similar way that Leighton and Rao [19] use their algorithm for sparsest cut on graphs to approximate balanced cut on graphs. In the case of symmetric submodular functions (a function  $f$  is symmetric if  $f(S) = f(\bar{S})$  for all  $S$ ), for a given  $b' \leq 1/3$ , we can find a  $b'$ -balanced cut whose cost is within a factor  $O\left(\frac{\gamma}{b-b'}\right)$  of the cost of any  $b$ -balanced cut, for  $b' < b \leq \frac{1}{2}$ . For arbitrary non-negative submodular functions, we can find a  $b'/2$ -balanced cut of cost within  $O\left(\frac{\gamma}{b-b'}\right)$  of any  $b$ -balanced cut, for any  $b'$  and  $b$  with  $b' < b \leq 1/2$ .

### 4 Submodular minimization with cardinality lower bound

We start with the lower bound result. Let  $R$  be a random subset of  $V$  of size  $\alpha = \frac{x\sqrt{n}}{5}$ , let  $\beta = \frac{x^2}{5}$ , and  $x$  be any parameter satisfying  $x^2 = \omega(\ln n)$ . We use the following two submodular functions:

$$\begin{aligned} f_1(S) &= \min(|S|, \alpha), \\ f_2(S) &= \min(\beta + |S \cap \bar{R}|, |S|, \alpha). \end{aligned} \quad (1)$$

**Lemma 4.1** *Any algorithm that makes a polynomial number of oracle queries has probability  $n^{-\omega(1)}$  of distinguishing the functions  $f_1$  and  $f_2$  above.*

**Proof.** By Lemma 2.1, it suffices to prove that for any set  $S$ , the probability that  $f_1(S) \neq f_2(S)$  is at most  $n^{-\omega(1)}$ . For the two functions above,  $\Pr[f_1(S) \neq f_2(S)]$  is maximized for sets  $S$  of size  $\alpha$ . And for a set  $S$  with  $|S| = \alpha$ ,  $f_1(S) \neq f_2(S)$  if and only if  $\beta + |S \cap \bar{R}| < |S|$ , or, equivalently,  $|S \cap R| > \beta$ . So we analyze the probability that  $|S \cap R| > \beta$ .

$R$  is a random subset of  $V$  of size  $\alpha$ . Let us consider a different set,  $R'$ , which is obtained by independently including each element of  $V$  with probability  $\alpha/n$ . The expected size of  $R'$  is  $\alpha$ , and the probability that  $|R'| = \alpha$  is at least  $1/(n+1)$ . Then

$$\begin{aligned} \Pr[|S \cap R| > \beta] &= \Pr[|S \cap R'| > \beta \mid |R'| = \alpha] \\ &\leq (n+1) \cdot \Pr[|S \cap R'| > \beta], \end{aligned}$$

and it suffices to show that  $\Pr[|S \cap R'| > \beta] = n^{-\omega(1)}$ . For this, we use Chernoff bounds [21].  $\mathbb{E}[|S \cap R'|] = \alpha|S|/n = \alpha^2/n = x^2/25$ , and  $\beta = 5 \cdot \mathbb{E}[|S \cap R'|]$ . So

$$\Pr[|S \cap R'| > \beta] < \left(\frac{e^4}{5^5}\right)^{\frac{\beta}{\mathbb{E}[|S \cap R'|]}} \leq 0.851x^2.$$

Since  $x^2 = \omega(\ln n)$ , this probability is  $n^{-\omega(1)}$ .  $\square$

**Theorem 4.2** *There is no  $(\rho, \sigma)$  bicriteria approximation algorithm for the SML problem, even with monotone functions, for any  $\rho$  and  $\sigma$  with  $\frac{\rho}{\sigma} = o\left(\sqrt{\frac{n}{\ln n}}\right)$ .*

### 4.1 Algorithm for SML

Our relaxed decision procedure for the SML problem with weights  $\{0, 1\}$  (Algorithm 3) builds up the solution out of multiple sets that it finds using submodular function minimization. If the weight requirement  $W$  is larger than half the total weight  $w(V)$ , then collecting sets whose ratio of function value to weight of new elements is low (less than  $2B/W$ ), until a total weight of at least  $W/2$  is collected, finds the required approximate solution. In the other case, if  $W$  is less than  $w(V)/2$ , the algorithm looks for sets  $T_i$  with low ratio of function value to the weight of new elements in the intersection of  $T_i$  and a random set  $S_i$ . These sets not only have small  $f(T_i)/w(T_i)$  ratio, but also have bounded function value  $f(T_i)$ . If such a set is found, then it is added to the solution.

---

**Algorithm 3** SML. Input:  $V, f, w : V \rightarrow \{0, 1\}, W, B, p$

---

```

1: Initialize  $U_0 = \emptyset; i = 0$ 
2: if  $W \geq w(V)/2$  then ▷ case  $W \geq \frac{w(V)}{2}$ 
3:   while  $w(U_i) < W/2$  do
4:     Let  $T_i \subseteq V$  minimize  $f(T) - \frac{2B}{W} \cdot w(T \setminus U_i)$ 
5:     if  $f(T_i) \geq \frac{2B}{W} \cdot w(T_i \setminus U_i)$  return fail
6:     else Let  $U_{i+1} = U_i \cup T_i; i = i + 1$ 
7:   end while
8:   return  $U = U_i$ 
9: end if
10: Let  $\alpha = \frac{2B}{W} \sqrt{\frac{n}{\ln n}}$  ▷ case  $W < \frac{w(V)}{2}$ 
11: while  $w(U_i) < W/2$  do
12:   Let  $S_i$  include each  $v \in V \setminus U_i$  with prob.  $\frac{W}{w(V)}$ 
13:   Let  $T_i \subseteq V$  minimize  $f(T) - \alpha \cdot w(T \cap S_i)$ 
14:   if  $f(T_i) \leq \alpha \cdot w(T_i \cap S_i)$  and  $f(T_i) \leq 4B\sqrt{\frac{n}{\ln n}}$ 
15:     then Let  $U_{i+1} = U_i \cup T_i; i = i + 1$ 
16:   if iterations exceed  $\frac{3n^{5/2}}{c} \ln\left(\frac{n}{1-p}\right)$ , return fail
17: end while
18: return  $U = U_i$ 

```

---

**Theorem 4.3** *Algorithm 3 is a  $(5\sqrt{\frac{n}{\ln n}}, \frac{1}{2})$  bicriteria decision procedure for the SML problem. That is, given a feasible instance, it outputs a set  $U$  with  $f(U) \leq 5\sqrt{\frac{n}{\ln n}}B$  and  $w(U) \geq W/2$  with probability at least  $p$ .*

**Proof.** Assume that the instance is feasible, and let  $U^* \subseteq V$  be a set with  $w(U^*) \geq W$  and  $f(U^*) < B$ . We consider two cases,  $W \geq w(V)/2$  and  $W < w(V)/2$ , which the algorithm handles separately.

First, assume that  $W \geq w(V)/2$  and consider one of the iterations of the while loop on line 3. By the loop condition,  $w(U_i) < W/2$ , so  $w(U^* \setminus U_i) > W/2$ . As a result, for the set  $U^*$ , the expression on line 4 is negative:

$$f(U^*) - \frac{2B}{W} \cdot w(U^* \setminus U_i) < f(U^*) - B < 0.$$

Then for the set  $T_i$  which minimizes this expression, it would also be negative, implying that  $w(T_i \setminus U_i)$  is positive, and so  $w(U_i)$  increases in each iteration. As a result, if the instance is feasible, then after at most  $n$  iterations of the loop on line 3, a set  $U$  is found with  $w(U) \geq W/2$ . For the function value, we have

$$f(U) \leq \sum_i f(T_i) < \frac{2B}{W} \sum_i w(T_i \setminus U_i) \leq \frac{2B}{W} w(V) \leq 4B$$

by our assumption about  $W$ .

The second case is  $W < w(V)/2$ . Assuming Claim 4.4 below, whose proof is omitted, we show that in each iteration of the while loop on line 11, with probability at least  $\frac{c}{3n^{3/2}}$ , a new non-empty set  $T_i$  is added to  $U$ . This implies that after  $\frac{3n^{5/2}}{c} \ln\left(\frac{n}{1-p}\right)$  iterations, the loop successfully terminates with probability at least  $p$ .

**Claim 4.4** *In each iteration of the loop on line 11 of Algorithm 3, the following hold with probability at least  $\frac{c}{3n^{3/2}}$ :*

$$w(U^* \cap S_i) > \frac{B}{\alpha} \quad \text{and} \quad w(\bar{U}^* \cap S_i) \leq 1.5W. \quad (2)$$

We show that if inequalities (2) hold, then the set  $T_i$  found by the algorithm on line 13 is non-empty and satisfies the conditions on line 14, which means that new elements are added to  $U$ . Since  $T_i$  is a minimizer of the expression on line 13, and using (2),

$$f(T_i) - \alpha \cdot w(T_i \cap S_i) \leq f(U^*) - \alpha \cdot w(U^* \cap S_i) < 0,$$

which means that  $T_i$  satisfies the first condition on line 14 and is non-empty. Moreover, from the same inequality and the second part of (2) we have

$$\begin{aligned} f(T_i) &\leq f(U^*) + \alpha \cdot (w(T_i \cap S_i) - w(U^* \cap S_i)) \\ &\leq B + \alpha \cdot w(\bar{U}^* \cap S_i) \leq B + 1.5\alpha W \leq 4B\sqrt{\frac{n}{\ln n}}, \end{aligned}$$

so  $T_i$  also satisfies the second condition on line 14.

Now we analyze the function value of the set output by the algorithm. Let  $T_i$  be the last set added to  $U$  by the while loop, and consider the set  $U_i$  just before  $T_i$  is added to it to produce  $U_{i+1}$ . By the loop condition, we have  $w(U_i) < W/2$ . Then, by submodularity and condition on line 14,

$$\begin{aligned} f(U_i) &\leq \sum_{j=0}^{i-1} f(T_j) \leq \sum_{j=0}^{i-1} \alpha \cdot w(T_j \cap S_j) \\ &\leq \alpha \cdot w(U_i) < \alpha \cdot \frac{W}{2} = B\sqrt{\frac{n}{\ln n}}. \end{aligned}$$

So for the set  $U$  that the algorithm outputs,  $f(U) \leq f(U_i) + f(T_i) \leq 5B\sqrt{\frac{n}{\ln n}}$ . And by the exiting condition of the while loop,  $w(U) \geq W/2$ .  $\square$

## 5 Submodular load balancing

For the lower bound, we give two monotone submodular functions that are hard to distinguish, but whose values of SLB optimal solutions differ by a large factor:

$$\begin{aligned} f_1(S) &= \min(|S|, \alpha), \\ f_2(S) &= \min\left(\sum_i \min(\beta, |S \cap V_i|), \alpha\right). \end{aligned}$$

Here  $\{V_i\}$  is a random (unknown to the algorithm) partition of  $V$  into  $m$  equal-sized sets. We set  $m = \frac{5\sqrt{n}}{x}$ ,  $\alpha = \frac{n}{m} = \frac{x\sqrt{n}}{5}$ ,  $\beta = \frac{x^2}{5}$ , with any  $x$  satisfying  $x^2 = \omega(\ln n)$ .

**Lemma 5.1** *Any algorithm that makes a polynomial number of oracle queries has probability  $n^{-\omega(1)}$  of distinguishing the functions  $f_1$  and  $f_2$  above.*

**Theorem 5.2** *The SLB problem is hard to approximate to a factor of  $o\left(\sqrt{\frac{n}{\ln n}}\right)$ .*

### 5.1 Algorithms for SLB

We note that the technique of Svitkina and Tardos [29] used for min-max multiway cut can be applied to the *non-uniform* SLB problem to obtain an  $O(\sqrt{n \log n})$  approximation algorithm, using the approximation algorithm for the SML problem presented in Section 4 as a subroutine. In this section we present two algorithms, with improved approximation ratios, for the *uniform* SLB problem. We begin by presenting a very simple algorithm that gives a  $\min(m, \lceil \frac{n}{m} \rceil) = O(\sqrt{n})$  approximation. Then we give a more complex algorithm that improves the approximation ratio to  $O\left(\sqrt{\frac{n}{\ln n}}\right)$ , thus matching the lower bound. Our first algorithm simply partitions the elements into  $m$  sets of roughly equal size.

**Theorem 5.3** *The algorithm that partitions the elements into  $m$  arbitrary sets of size at most  $\lceil \frac{n}{m} \rceil$  each is a  $\min(m, \lceil \frac{n}{m} \rceil)$  approximation for the SLB problem.*

**Proof.** Let  $\{U_1^*, \dots, U_m^*\}$  denote the optimal solution with value  $B$ , and let  $A$  be the value of the solution  $\{S_1, \dots, S_m\}$  found by the algorithm. We exhibit two lower bounds on  $B$  and two upper bounds on  $A$ , and then establish the approximation ratio by comparing these bounds. For the lower bounds on  $B$ , we claim that  $B \geq \max_{j \in V} f(\{j\})$  and  $B \geq f(V)/m$ . For the first one, let  $j$  be the element

maximizing  $f(\{j\})$ , and let  $U_i^*$  be the set in the optimal solution that contains  $j$ . Then  $B \geq f(U_i^*) \geq f(\{j\})$  by monotonicity. For the second bound, by submodularity we have that  $f(V) \leq \sum_i f(U_i^*) \leq mB$ . To bound  $A$ , we notice that  $A \leq f(V)$  (by monotonicity), and that  $A \leq \lceil \frac{n}{m} \rceil \max_{j \in V} f(\{j\})$ , since each set  $S_i$  contains at most  $\lceil \frac{n}{m} \rceil$  elements, and  $f(S_i) \leq \sum_{j \in S_i} f(\{j\})$ . Comparing with the lower bounds on  $B$ , we get the result.  $\square$

---

**Algorithm 4** SLB. Input:  $V, m > \sqrt{\frac{n}{\ln n}}, f, B, p$

---

- 1: **if** for any  $v \in V, f(\{v\}) \geq B$ , **return fail**
  - 2: Let  $\alpha = Bm/\sqrt{n \ln n}$ ; Initialize  $V' = V, i = 0$
  - 3: **while**  $|V'| > m\sqrt{\frac{n}{\ln n}}$  **do**
  - 4:   Let  $S$  include each  $v \in V'$  indep. with prob.  $\frac{n}{m|V'|}$
  - 5:   **if**  $|S| \leq 2\frac{n}{m}$  **then**
  - 6:     Let  $T \subseteq S$  minimize  $f(T) - \alpha \cdot |T|$
  - 7:     **if**  $f(T) - \alpha \cdot |T| < 0$
  - 8:       **then** Set  $T_i = T; i = i + 1; V' = V' \setminus T$
  - 9:   **end if**
  - 10:   **if** iterations exceed  $\frac{2m^2}{c} \ln(\frac{n}{1-p})$ , **return fail**
  - 11: **end while**
  - 12: Let  $\mathcal{T}$  be collection of sets  $T_i$  produced by while loop
  - 13: Partition  $\mathcal{T}$  into  $m$  groups  $\mathcal{T}_1, \dots, \mathcal{T}_m$ , such that  $\sum_{i: T_i \in \mathcal{T}_j} |T_i| \leq 3\frac{n}{m}$  for each  $\mathcal{T}_j$
  - 14: Let  $U_1, \dots, U_m$  be any partition of  $V'$  with each set of size at most  $\sqrt{\frac{n}{\ln n}}$
  - 15: For each  $j \in \{1, \dots, m\}$ , let  $V_j = U_j \cup \bigcup_{T_i \in \mathcal{T}_j} T_i$
  - 16: **return**  $\{V_1, \dots, V_m\}$
- 

For the more complex Algorithm 4, we assume that  $m > \sqrt{\frac{n}{\ln n}}$ , because for lower values of  $m$  the above simple algorithm gives the desired approximation. Also, the simple algorithm has better guarantee for all  $n \leq e^{16}$ , so when analyzing Algorithm 4, we can assume that  $n$  is sufficiently large for certain inequalities to hold, such as  $\ln^3 n < n$ . The algorithm finds small disjoint sets of elements that have low ratio of function value to size. Once a sufficient number of elements is grouped into such low-ratio sets, these sets are combined to form  $m$  final sets of the partition, while adding a few remaining elements. These final sets have roughly  $n/m$  elements each, so using submodularity and the low ratio property, we can bound the function value for each set in the partition.

First we describe how some of the steps of algorithm work. The loop condition  $|V'| > m\sqrt{\frac{n}{\ln n}}$  and our assumptions  $m > \sqrt{\frac{n}{\ln n}}$  and  $\ln^3 n < n$  imply that the probability  $\frac{n}{m|V'|}$  (used on line 4) is less than one. The partition on line 14 can be found because at this point, the size of  $V'$  is at most  $m\sqrt{\frac{n}{\ln n}}$ . For the partitioning done on line 13, we note that since each  $T_i$  is a subset of a sample set  $S$  with  $|S| \leq 2n/m$ , it holds that  $|T_i| \leq 2n/m$ . Also, the

total number of elements contained in all sets  $T_i$  is at most  $n$  (since they are disjoint). So a simple greedy procedure that adds the sets  $T_i$  to  $\mathcal{T}_j$  in arbitrary order, until the total number of elements is at least  $n/m$ , will produce at most  $m$  groups, each with at most  $3n/m$  elements.

**Theorem 5.4** *If given a feasible instance of the SLB problem, Algorithm 4 outputs a solution of value at most  $4\sqrt{\frac{n}{\ln n}} \cdot B$  with probability at least  $p$ .*

**Proof.** By monotonicity of  $f$ , the algorithm exits on line 1 only if the instance is infeasible. Assume that the instance is feasible and let  $\{U_1^*, \dots, U_m^*\}$  denote the solution with  $\max_j f(U_j^*) < B$ . We consider one iteration of the while loop and show that with probability at least  $\frac{c}{2n}$  it finds a set  $T \subseteq S$  satisfying  $f(T) - \alpha \cdot |T| < 0$ . Then the probability that the size of  $V'$  is reduced to  $m\sqrt{\frac{n}{\ln n}}$  after  $\frac{2m^2}{c} \ln(\frac{n}{1-p})$  iterations is at least  $p$ .

Assume, without loss of generality, that  $U_1^*$  is the set that maximizes  $|U_j^* \cap V'|$  for this iteration of the loop. If we let  $n' = |V'|$ , then  $|U_1^* \cap V'| \geq n'/m$ . Suppose the sample  $S$  found by the algorithm has size at most  $2n/m$ , and let  $t = |U_1^* \cap S|$  denote the size of the overlap of  $S$  and  $U_1^*$ . By monotonicity of the function  $f$ , we know that  $f(U_1^* \cap S) \leq f(U_1^*) < B$ . Since the algorithm finds a set  $T \subseteq S$  minimizing  $f(T) - \alpha|T|$ , we know that the value of this expression for  $T$  is at most that for  $U_1^* \cap S$ :

$$f(T) - \alpha|T| \leq f(U_1^* \cap S) - \alpha|U_1^* \cap S| < B - \frac{Bmt}{\sqrt{n \ln n}}.$$

In order to have  $f(T) - \alpha|T| < 0$ , we need  $t \geq \frac{\sqrt{n \ln n}}{m}$ . But the probability that both  $t \geq \frac{\sqrt{n \ln n}}{m}$  and  $|S| \leq 2n/m$  is at least  $\frac{c}{2n}$  (proof omitted).

This establishes that on feasible instances, the algorithm successfully terminates with probability at least  $p$ . Let us now consider the function value on any of the sets  $V_j$  output by the algorithm. By submodularity,

$$f(V_j) \leq \sum_{v \in U_j} f(\{v\}) + \sum_{T_i \in \mathcal{T}_j} f(T_i).$$

For each  $T_i$  we know that  $f(T_i) < \alpha \cdot |T_i|$ , and by the check performed on line 1, we have  $f(\{v\}) < B$  for each  $v \in V$ . Using this and the bounds on set sizes,

$$\begin{aligned} f(V_j) &\leq B\sqrt{\frac{n}{\ln n}} + \alpha \sum_{T_i \in \mathcal{T}_j} |T_i| \leq B\sqrt{\frac{n}{\ln n}} + \alpha \cdot \frac{3n}{m} \\ &= B \cdot \left( \sqrt{\frac{n}{\ln n}} + \frac{m}{\sqrt{n \ln n}} \frac{3n}{m} \right) = 4\sqrt{\frac{n}{\ln n}} \cdot B. \end{aligned}$$

$\square$

## 6 Learning submodular functions

We present a lower bound for the problem of learning submodular functions, which holds even for the special case of monotone functions. We use the same functions (1) as for the SML lower bound, observing that they are monotone.

**Theorem 6.1** *Any algorithm that makes a polynomial number of oracle queries cannot approximate the problem of learning monotone submodular functions to a factor  $o\left(\sqrt{\frac{n}{\ln n}}\right)$ .*

### 6.1 Learning monotone 2P functions

Recall that a 2P function is one for which there is a set  $R \subseteq V$  such that the function value  $f(S)$  depends only on  $|S \cap R|$  and  $|S \cap \bar{R}|$ . Our algorithm for learning monotone 2P functions (Algorithm 5) uses the following observation.

**Lemma 6.2** *Given two sets  $S$  and  $T$  such that  $|S| = |T|$ , but  $f(S) \neq f(T)$ , a 2P function can be learned exactly using a polynomial number of oracle queries.*

**Proof.** This is done by inferring what the set  $R$  is. Using  $S$  and  $T$ , we find two sets which differ by exactly one element and have different function values. Fix an ordering of the elements of  $S$ ,  $\{s_1, \dots, s_k\}$ , and an ordering of elements of  $T$ ,  $\{t_1, \dots, t_k\}$ , such that the elements of  $S \cap T$  appear last in both orderings, and in the same sequence. Let  $S_0 = S$ , and  $S_i$  be the set  $S$  with the first  $i$  elements replaced by the first  $i$  elements of  $T$ :  $S_i = \{t_1, \dots, t_i, s_{i+1}, \dots, s_k\}$ . Evaluate  $f$  on each of the sets  $S_i$  in order, until the first time that  $f(S_{i-1}) \neq f(S_i)$ . Such an  $i$  must exist since  $S_k = T$ , and by assumption  $f(T) \neq f(S)$ . Let  $U = \{t_1, \dots, t_{i-1}, s_{i+1}, \dots, s_k\}$ , so that  $S_{i-1} = U \cup \{s_i\}$  and  $S_i = U \cup \{t_i\}$ .

The fact that  $f(U \cup \{s_i\}) \neq f(U \cup \{t_i\})$  tells us that either  $s_i \in R$  and  $t_i \notin R$ , or vice versa. Without loss of generality, we assume the former (since the names of  $R$  and  $\bar{R}$  can be interchanged). Now all elements in  $V \setminus U$  can be classified as belonging or not belonging to  $R$ . In particular, if for some element  $j \in \bar{U}$ ,  $f(U \cup \{j\}) = f(U \cup \{s_i\})$ , then  $j \in R$ ; otherwise  $f(U \cup \{j\}) = f(U \cup \{t_i\})$ , and  $j \notin R$ . To test an element  $u \in U$ , evaluate  $f(U - \{u\} + \{s_i, t_i\})$ . This is the set  $S_{i-1}$  with element  $u$  replaced by  $t_i$ . If  $u \in \bar{R}$ , then replacing one element from  $\bar{R}$  by another will have no effect on the function value, and it will be equal to  $f(S_{i-1})$ . If  $u \in R$ , then we have replaced an element from  $R$  by an element from  $\bar{R}$ , and we know that this changes the function value to  $f(S_i)$ . So all elements of  $V$  can be tested for their membership in  $R$ , and then all function values can be learned by querying sets  $W \subseteq V$  with all possible values of  $|W \cap R|$  and  $|W \cap \bar{R}|$ .  $\square$

---

**Algorithm 5** Learning monotone 2P function. Input:  $V, f, p$

---

- 1: Query values of  $f(\emptyset)$ ,  $f(V)$ , and  $f(\{j\})$  for each  $j \in V$
- 2: For each  $i \in \{2, \dots, n-1\}$ , independently generate  $n^{10} \ln\left(\frac{4n}{1-p}\right)$  random sets by including each element of  $V$  into each set with probability  $\frac{i}{n}$ . Query the function value for each of these sets.
- 3: If the previous two steps produce any two sets  $S_1$  and  $S_2$  with  $|S_1| = |S_2|$  and  $f(S_1) \neq f(S_2)$ , then learn the function exactly, as described in Lemma 6.2.
- 4: Else, let  $j \in V$  be an arbitrary element, and output

$$\hat{f}(S) = \begin{cases} f(\emptyset) & \text{if } S = \emptyset \\ f(\{j\}) & \text{if } 1 \leq |S| \leq 2\sqrt{n} \\ \frac{f(V)}{2\sqrt{n}} & \text{if } |S| > 2\sqrt{n} \end{cases}$$


---

**Theorem 6.3** *With probability at least  $p$ , the function  $\hat{f}$  returned by Algorithm 5 satisfies  $\hat{f}(S) \leq f(S) \leq 2\sqrt{n} \cdot \hat{f}(S)$  for all sets  $S \subseteq V$ .*

**Proof.** If the algorithm finds two sets  $S_1$  and  $S_2$  such that  $|S_1| = |S_2|$  and  $f(S_1) \neq f(S_2)$  during the sampling stage (steps 1 and 2), then the correctness of the output is implied by Lemma 6.2. If it does not find such sets, then it outputs the function  $\hat{f}$  shown in step 4. It obviously satisfies the inequality for the case that  $S = \emptyset$ . For the case that  $1 \leq |S| \leq 2\sqrt{n}$ , we observe that if the algorithm reaches step 4, it must be that the value of  $f$  is identical for all singleton sets, i.e.  $f(\{j\}) = f(\{j'\})$  for all  $j, j' \in V$ . Now,  $f(S) \geq f(\{j\}) = \hat{f}(S)$  by monotonicity. Also, by submodularity,  $f(S) \leq \sum_{j \in S} f(\{j\}) = |S| \cdot \hat{f}(S) \leq 2\sqrt{n} \cdot \hat{f}(S)$ , establishing the correctness for the case that  $|S| \leq 2\sqrt{n}$ . For the last case,  $|S| > 2\sqrt{n}$ , the inequality  $f(S) \leq f(V) = 2\sqrt{n} \cdot \hat{f}(S)$  follows by monotonicity. For the other one,  $\hat{f}(S) \leq f(S)$ , we need an additional nontrivial lemma.

Since the 2P function  $f(S)$  depends only on two values,  $|S \cap R|$  and  $|S \cap \bar{R}|$ , let us denote by  $f(k, l)$  the value of the function  $f$  on a set  $S$  with  $|S \cap R| = k$  and  $|S \cap \bar{R}| = l$ . We say that such a set  $S$  corresponds to the pair  $(k, l)$ . We assume that  $0 < |R| < n$ , because if  $|R| = 0$  or  $|R| = n$ , then  $f(S)$  is a function that depends only on  $|S|$ , and it equally well can be represented as a 2P function with any other set  $\bar{R}$ . Furthermore, we assume without loss of generality that  $|R| \leq |\bar{R}|$  (otherwise interchange  $R$  and  $\bar{R}$ ), and let  $K = |R|$  and  $L = |\bar{R}|$  (which are not known to the algorithm).

**Lemma 6.4** *For any  $k$  and any  $l$ ,  $f(k, 0) \geq \frac{k}{2n} f(V)$  and  $f(0, l) \geq \frac{l}{2n} f(V)$ .*

Using this lemma to finish the proof, let  $k = |S \cap R|$  and  $l = |S \cap \bar{R}|$ . By monotonicity,  $f(S) \geq f(k, 0)$  and

$f(S) \geq f(0, l)$ . Moreover, since  $|S| = k + l \geq 2\sqrt{n}$ , we have  $\max(k, l) \geq \sqrt{n}$ . So, using Lemma 6.4,  $f(S) \geq \frac{\max(k, l)}{2n} f(V) \geq \frac{f(V)}{2\sqrt{n}} = \hat{f}(S)$ .  $\square$

The proof of Lemma 6.4 is involved, and we sketch the main ideas here. We call a pair  $(k, l)$  *balanced* if  $k/l$  is close to  $K/L$ . Then, with significant probability, the algorithm samples sets corresponding to all balanced pairs. Since the algorithm checks for sets of the same size with different function values, we can assume that if it proceeds to step 4, then for sets  $S$  corresponding to balanced pairs,  $f(S)$  is a function  $F$  that depends only on  $|S|$ . We use submodularity to show that  $F$  is concave. Then we decompose  $f(k, 0)$  as  $\sum_{i=1}^k [f(i, 0) - f(i-1, 0)]$  and lower-bound each term in this sum separately by comparing it to an increment  $f(i, j) - f(i-1, j)$  for some  $j$  with  $(i, j)$  balanced. Then, using concavity of  $F$ , we lower-bound their sum.

## 7 Acknowledgements

We thank Mark Sandler for his help with some of the calculations, the anonymous referees for their comments and suggestions, and the authors of [8] for making their manuscript publicly available.

## References

- [1] S. Arora, E. Hazan, and S. Kale.  $O(\sqrt{\log n})$  approximation to sparsest cut in  $\tilde{O}(n^2)$  time. In *Proc. 45th IEEE Symp. on Foundations of Computer Science*, pages 238–247, 2004.
- [2] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proc. 36th ACM Symp. on Theory of Computing*, 2004.
- [3] C. Chekuri, G. Calinescu, M. Pal, and J. Vondrak. Maximizing a submodular set function subject to a matroid constraint. In *IPCO*, pages 182–196, 2007.
- [4] W. Cunningham. Minimum cuts, modular functions, and matroid polyhedra. *Networks*, 15:205–215, 1985.
- [5] U. Feige, V. Mirrokni, and J. Vondrak. Maximizing non-monotone submodular functions. In *FOCS*, 2007.
- [6] S. Fujishige. Polymatroid dependence structure of a set of random variables. *Info. and Control*, 39:55–72, 1978.
- [7] G. Gens and E. Levner. Computational complexity of approximation algorithms for combinatorial problems. In *Proc. 8th Intl. Symp. on Math. Foundations of Comput. Sci.* Lecture Notes in Comput. Sci. 74, Springer-Verlag, 1979.
- [8] M. Goemans, N. Harvey, R. Kleinberg, and V. Mirrokni. On learning submodular functions. Manuscript, 2008.
- [9] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.
- [10] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.
- [11] N. Harvey. *Matchings, Matroids and Submodular Functions*. PhD thesis, MIT, 2008.
- [12] A. Hayrapetyan, D. Kempe, M. Pal, and Z. Svitkina. Unbalanced graph cuts. In *Proc. 13th European Symposium on Algorithms*, 2005.
- [13] A. Hayrapetyan, C. Swamy, and E. Tardos. Network design for information networks. In *Proc. 16th ACM Symp. on Discrete Algorithms*, pages 933–942, 2005.
- [14] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. ACM*, 34:144–162, 1987.
- [15] B. Hoppe and É. Tardos. The quickest transshipment problem. *Math. Oper. Res.*, 25(1):36–62, 2000.
- [16] S. Iwata. A faster scaling algorithm for minimizing submodular functions. *SIAM J. Comput.*, 32:833–840, 2003.
- [17] S. Iwata. Submodular function minimization. *Math. Programming*, 112:45–64, 2008.
- [18] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.
- [19] F. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46, 1999.
- [20] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming*, 46:259–271, 1990.
- [21] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1990.
- [22] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.
- [23] J. B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. In *IPCO*, 2007.
- [24] M. Queyranne. Minimizing symmetric submodular functions. *Math. Programming*, 82:3–12, 1998.
- [25] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. 40th ACM Symp. on Theory of Computing*, pages 255–263, 2008.
- [26] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. of Combinatorial Theory, Ser. B*, 80(2):346–355, 2000.
- [27] M. Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.
- [28] Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. arXiv:0805.1071 [cs.DS], 2008.
- [29] Z. Svitkina and E. Tardos. Min-max multiway cut. In *Proc. 7th APPROX*, pages 207–218, 2004.
- [30] Z. Svitkina and E. Tardos. Facility location with hierarchical facility costs. In *Proc. 17th ACM Symp. on Discrete Algorithms*, pages 153–161, 2006.
- [31] C. Swamy, Y. Sharma, and D. Williamson. Approximation algorithms for prize collecting steiner forest problems with submodular penalty functions. In *Proc. 18th ACM Symp. on Discrete Algorithms*, pages 1275–1284, 2007.
- [32] L. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.
- [33] L. Zhao, H. Nagamochi, and T. Ibaraki. Greedy splitting algorithms for approximating multiway partition problems. *Mathematical Programming*, 102(1):167–183, 2005.