RIMS Kôkyûroku Bessatsu
B23 (2010), 213–232

Algorithms and Lower Bounds for Submodular Cuts and Approximating Submodular Functions

By

Zoya Svitkina[*] and Lisa Fleischer[**]

Abstract

We study the *submodular sparsest cut* problem, which is a generalization of the classical sparsest cut problem obtained by replacing the graph cut function with a general submodular function, and establish matching upper and lower bounds for its approximability. Then we apply the approximation algorithm for submodular sparsest cut to obtain bicriteria approximation results for a related problem, *submodular balanced cut*, which generalizes the balanced cut problem on graphs.

We also give an improved lower bound for the problem of approximating a monotone submodular function everywhere. Then we present an algorithm for approximating monotone submodular functions with special structure, called two-partition functions. This algorithm's guarantee is close to the lower bound, which uses two-partition functions, and therefore applies even in this special case.

§ 1. Introduction

A function f defined on subsets of a ground set V is called *submodular* if for all subsets $S, T \subseteq V$, it satisfies the inequality $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$. 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 study the *submodular sparsest cut* and *submodular balanced cut* problems, which generalize their respective graph cut problems. In particular, they are obtained by replacing graph cut functions in the objectives of the classical problems

Received September 16, 2008. Revised June 20, 2009.
2000 Mathematics Subject Classification(s): 68Q25
This paper summarizes and extends some of the results that have appeared in [24]. The work was supported in part by NSF grant CCF-0728869.
[*]Department of Computing Science, University of Alberta, Canada.
[**]Department of Computer Science, Dartmouth College, USA.

with arbitrary submodular functions. Another problem that we study is *approximating a submodular function everywhere*, which was recently introduced by Goemans, Harvey, Iwata, and Mirrokni [6, 9]. All of these problems are defined on a set V of n elements with a nonnegative submodular function $f : 2^V \to \mathbb{R}_{\geq 0}$. 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 of elements $\{\{u_i, v_i\} \mid u_i, v_i \in V\}$, each with a demand $d_i \geq 0$, 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 $(S, \bar{S})$[1]. A special case of the SSC problem is the *weighted* SSC problem, in which each element $v \in V$ 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 $w : V \to \mathbb{R}_{\geq 0}$, 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 $f(S)$. A special case is the balanced cut problem for *symmetric* submodular functions, which are functions that satisfy $f(S) = f(\bar{S})$ for all sets $S \subseteq V$.

Approximating a Submodular Function Everywhere: Produce a function $\hat{f}$ (not necessarily submodular) that for all sets $S \subseteq V$ 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 consider the special case of monotone two-partition functions, defined as follows. A submodular function f on a ground set V 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 $|S \cap \bar{R}|$. A function f is *monotone* if $f(S) \leq f(T)$ whenever $S \subseteq T$.

§ 1.1. Motivation

Submodular functions arise in a variety of contexts, often in optimization settings. The submodular sparsest and balanced cut problems considered in this paper use submodular functions to generalize two well-studied problems in computer science. These generalizations capture many variants of their corresponding classical problems. For example, they 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.

[1]For any set $S \subseteq V$, we use $\bar{S}$ to denote its complement set, $V \setminus S$.

§ 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 [7, 8]. Further research has yielded several more combinatorial approaches [12, 13, 14, 19, 20, 22].

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 constraints [3, 5, 15, 18, 23]. Research on other optimization problems has also used submodular functions or their minimization, including [10, 11, 25, 26, 27, 28].

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 everywhere with a polynomial number of value oracle queries. Some lower bounds on the approximation guarantees achievable in this model are given in [6, 9].

The sparsest and balanced cut problems are NP-hard even on graphs. 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 $O(\log n)$ [21].

§ 1.3. Our results and techniques

We establish matching upper and lower bounds for the approximability of the submodular sparsest cut problem. Surprisingly, these factors are quite high, of the order of $\sqrt{\frac{n}{\ln n}}$. 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, 6]. To prove the upper bound, we present a randomized approximation algorithm which samples a random subset of the ground set, and assigns weights to elements based on this random set and the input demands. Then, based on these weights, it uses submodular function minimization to find a candidate solution. We show that with relatively high probability (inverse polynomial), a sample is obtained which separates a higher than expected fraction of demand separated by the optimal solution. And if this is the case, then the obtained solution satisfies the algorithm's guarantee.

For submodular balanced cut, we also show an approximation lower bound of $\Omega\left(\sqrt{\frac{n}{\ln n}}\right)$. Then we use the algorithm for weighted SSC as a subroutine to obtain two bicriteria approximation algorithms in a similar way as Leighton and Rao [16] do for graphs. Our first algorithm works for symmetric submodular functions, and for a given $b' \leq 1/3$, it finds 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 $b' < b \leq \frac{1}{2}$. The second algorithm works for arbitrary non-negative submodular functions and produces 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 $b' < b \leq 1/2$.

For approximating monotone submodular functions everywhere, 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 [6], and matches the lower bound for arbitrary submodular functions, also in [6]. Our lower bound proof for this problem, as well as those in [6], use 2P functions, and thus still hold for this special case. We show that 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 approximation 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 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 all subsets $T \subseteq V$) expressions like $f(T) - \alpha \cdot |T \cap S|$, where α is a constant and S is a fixed subset of V.

We present our algorithm for the SSC problem by providing a *randomized relaxed decision procedure* for it. 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 γB, where γ 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 γ-approximate solution with probability at least p. On an infeasible instance, either of the two outcomes is allowed. Our procedure runs in time polynomial in n and $\ln \frac{1}{1-p}$, and can be turned into a randomized approximation algorithm by finding upper and lower bounds for the optimum and performing binary search.

We say that an algorithm *distinguishes* two functions f_1 and f_2 if the output that it produces upon receiving (an oracle for) the function f_1 as input is different than the output that it produces upon receiving the function f_2 as input. 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. □

§ 3. Lower bounds for submodular sparsest and balanced cuts

To show the lower bounds for submodular sparsest and balanced cuts, we use the following result from [6], whose proof we present for completeness, and then apply Lemma 2.1.

Lemma 3.1. **(Goemans et al. [6])** *Fix an arbitrary subset $S \subseteq V$, and then let R be a random subset of V of size $n/2$. Then for any ε such that $\varepsilon^2 = \frac{1}{n} \cdot \omega(\ln n)$, for $\beta = \frac{n}{4}(1+\varepsilon)$, and for the submodular[2] functions*

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

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

[2]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, then adding a to S increases the function value by at most x.

Proof. Since $f_1 \geq f_2$ for all sets, the two functions differ on S if and only if $f_2(S) - f_1(S) < 0$. First, we claim that the probability $\Pr[f_2(S) - f_1(S) < 0]$ is maximized when $|S| = \frac{n}{2}$. For this, suppose that $|S| \geq \frac{n}{2}$. Then

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

This quantity is negative whenever $\beta + |S \cap \bar{R}| - \frac{n}{2}$ is negative. But this expression can only decrease if an element is removed from S, so we conclude that in the range $|S| \geq n/2$, the two functions are most likely to be different when $|S| = n/2$.

A similar argument can be made for the case of $|S| \leq n/2$. In that case,

$$f_2(S) - f_1(S) \;=\; \min\left(\beta + |S \cap \bar{R}|,\ |S|\right) - |S|,$$

and this expression is negative whenever $\beta + |S \cap \bar{R}| - |S| = \beta - |S \cap R| < 0$. This expression can only decrease if an element is added to S, so again the probability of f_1 and f_2 having different values is maximized when $|S| = n/2$.

We now show that for a set S of size $n/2$, the probability that $f_1(S) \neq f_2(S)$ is low. This probability is

$$\begin{aligned}\Pr\left[f_1(S) \neq f_2(S)\right] &= \Pr\left[\min\left(\beta + |S \cap \bar{R}|,\ |S|,\ \frac{n}{2}\right) < \min\left(|S|,\ \frac{n}{2}\right)\right] \\ &= \Pr\left[\beta + |S \cap \bar{R}| < \frac{n}{2}\right] \;=\; \Pr\left[|S \cap \bar{R}| < \frac{n}{4}(1-\varepsilon)\right].\end{aligned}$$

If instead of choosing R as a random subset of V of size $n/2$, we consider a set R' for which each element is chosen independently with probability $1/2$, the probability that we are interested in becomes

$$\begin{aligned}\Pr\left[f_1(S) \neq f_2(S)\right] &= \Pr\left[|S \cap \bar{R}'| < \frac{n}{4}(1-\varepsilon)\ \middle|\ |R'| = \frac{n}{2}\right] \\ &= \frac{\Pr\left[|S \cap \bar{R}'| < \frac{n}{4}(1-\varepsilon) \wedge |R'| = \frac{n}{2}\right]}{\Pr\left[|R'| = \frac{n}{2}\right]} \\ &\leq (n+1) \cdot \Pr\left[|S \cap \bar{R}'| < \frac{n}{4}(1-\varepsilon)\right].\end{aligned}$$

This allows us to make a switch to independent variables, so that we can use Chernoff bounds [17]. The expectation μ of $|S \cap \bar{R}'|$ is equal to $|S|/2 = n/4$, so

$$\Pr\left[|S \cap \bar{R}'| < (1-\varepsilon)\mu\right] \;<\; e^{-\mu\varepsilon^2/2} \;=\; e^{-\omega(\ln n)} \;=\; n^{-\omega(1)},$$

since $\varepsilon^2 = \frac{1}{n} \cdot \omega(\ln n)$. This gives $\Pr\left[f_1(S) \neq f_2(S)\right] < (n+1) \cdot n^{-\omega(1)} = n^{-\omega(1)}$. □

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 use these results to establish the hardness of the 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 the objective function value on this set.

Theorem 3.3. *The 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 γ-approximation algorithm for the 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 $\varepsilon^2 = \frac{1}{n} \cdot \omega(\ln n)$, and consider the uniform demands, with $d = 1$ for any pair of distinct elements. If the algorithm is given the function f_2 of Lemma 3.1 as input, then with high probability it has to output a set S with ratio $\frac{f_2(S)}{|S|\cdot|\bar{S}|} \le \frac{\gamma\varepsilon}{n} = \frac{1}{2n}$, since the optimal uniform sparsest cut for f_2, achieved by the set R, has ratio $\frac{\beta - n/4}{n^2/4} = \frac{\varepsilon}{n}$. However, for the function f_1, all non-empty sets have ratio $\frac{f_1(S)}{|S|\cdot|\bar{S}|} = \frac{1/2}{\max(|S|,|\bar{S}|)} > \frac{1}{2n}$. So if the algorithm is given f_1 as input, its output value differs from the case of f_2, contradicting Corollary 3.2.

A very similar proof establishes the lower bound for submodular balanced cut. □

§ 4. Algorithm for submodular sparsest cut

Our approach for solving SSC uses a random set S to assign weights to nodes (see Algorithm 1). 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. We begin by presenting a technical lemma about random sampling that is used for bounding probabilities in the analysis of the algorithm. We use the constant $c = 1/(e^2\sqrt{2\pi})$ throughout.

Lemma 4.1. *Suppose that m elements are selected independently, with probability q each. Then for $\varepsilon < \frac{1-q}{q}$, the probability that $qm(1+\varepsilon)$ elements are selected is at least $\frac{c}{\sqrt{m}} \exp\left[\frac{-\varepsilon^2 qm}{1-q}\right]$.*

Proof. For convenience, let $\kappa = q(1+\varepsilon)$. Using an approximation that $\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \le n! \le e\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, which is derived from Stirling's formula, we obtain the bound

$$\binom{m}{m\kappa} = \frac{m!}{(m\kappa)!(m-m\kappa)!} \ge \frac{\sqrt{2\pi}}{(e\sqrt{2\pi})^2} \cdot \frac{\sqrt{m}}{\sqrt{m\kappa}\sqrt{m-m\kappa}} \cdot \frac{(m/e)^m}{(m\kappa/e)^{m\kappa}((m-m\kappa)/e)^{m-m\kappa}}$$
$$\ge \frac{1}{e^2\sqrt{2\pi}} \cdot \frac{1}{\sqrt{m}} \cdot \frac{1}{\kappa^{m\kappa}(1-\kappa)^{m-m\kappa}}.$$

Then the desired probability is

$$\begin{aligned}\binom{m}{m\kappa} q^{m\kappa}(1-q)^{m-m\kappa} &\geq c\cdot m^{-\frac{1}{2}}\cdot\frac{q^{m\kappa}\cdot(1-q)^{m-m\kappa}}{\kappa^{m\kappa}\cdot(1-\kappa)^{m-m\kappa}}\\ &= c\cdot m^{-\frac{1}{2}}\cdot\left(\frac{1}{1+\varepsilon}\right)^{m\kappa}\cdot\left(\frac{1-q}{1-q(1+\varepsilon)}\right)^{m-m\kappa}\\ &= c\cdot m^{-\frac{1}{2}}\cdot\frac{1}{(1+\varepsilon)^{m\kappa}}\cdot\frac{1}{\left(1-\frac{\varepsilon q}{1-q}\right)^{m-m\kappa}}\\ &\geq c\cdot m^{-\frac{1}{2}}\cdot\exp\left[-\varepsilon m\kappa+\frac{\varepsilon q}{1-q}m(1-\kappa)\right],\end{aligned}$$

where we have used the fact that $1+x\leq e^x$ for all x. The assumption that $\varepsilon<\frac{1-q}{q}$ ensures that the second denominator is positive. Now, the exponent of e is equal to

$$-\varepsilon qm(1+\varepsilon)+\frac{\varepsilon q}{1-q}m(1-q-\varepsilon q) \;=\; -\varepsilon qm-\varepsilon^2 qm+\varepsilon qm-\frac{\varepsilon^2q^2m}{1-q} \;=\; \frac{-\varepsilon^2 qm}{1-q},$$

concluding the proof. □

Algorithm 1 Submodular sparsest cut. Input: V, f, d, B, p

```
1: for $\frac{4n^2}{c}\ln(\frac{1}{1-p})$ iterations do
2:     Choose a random set $S$ by including each $v \in V$ independently with prob. $\frac{1}{2}$
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:         Let $s_i \in \{u_i, v_i\} \cap S$ and $t_i \in \{u_i, v_i\} \setminus S$ ▷ name the unique node in each set
6:         Update weights $w(s_i) \leftarrow w(s_i) + d_i$;  $w(t_i) \leftarrow w(t_i) - d_i$
7:     end for
8:     Let $\alpha = 4\sqrt{\frac{n}{\ln n}} \cdot B$
9:     Let $T$ be a subset of $V$ minimizing $f(T) - \alpha \cdot \sum_{v \in T} w(v)$
10:    if $f(T) - \alpha \cdot \sum_{v \in T} w(v) < 0$, return $T$
11: end for
12: return fail
```

The following lemma shows that any set T satisfying the condition on line 10 of the algorithm is a solution that satisfies the algorithm's approximation guarantee.

Lemma 4.2. *If for some $T\subseteq V$, it holds that $f(T)-\alpha\cdot\sum_{v\in T}w(v)<0$, then*

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

Proof. $\sum_{v\in T} w(v)$ is equal to

$$\sum_{i:s_i\in T} d_i - \sum_{i:t_i\in T} d_i \;=\; \sum_{i:s_i\in T, t_i\notin T} d_i - \sum_{i:t_i\in T, s_i\notin T} d_i \;\le\; \sum_{i:s_i\in T, t_i\notin T} d_i \;\le\; \sum_{i:|T\cap\{u_i,v_i\}|=1} d_i.$$

Now using the assumption of the lemma we have

$$f(T) - \alpha \sum_{i:|T\cap\{u_i,v_i\}|=1} d_i \;\le\; f(T) - \alpha\sum_{v\in T} w(v) \;<\; 0. \tag{4.1}$$

Since the function f is non-negative, it must be that $\sum_{i:|T\cap\{u_i,v_i\}|=1} d_i > 0$. Rearranging the terms, we get $f(T)/\sum_{i:|T\cap\{u_i,v_i\}|=1} d_i < \alpha$. □

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

Proof. Let $\varepsilon = \sqrt{\frac{\ln n}{n}}$. We denote by A the event that $|U^*\cap S| \ge \frac{m}{2}(1+\varepsilon)$, where S is the random set chosen by Algorithm 1, and bound the above probability by the following product:

$$\Pr\left[\sum_{v\in U^*} w(v) \ge \frac{\varepsilon}{4}D^*\right] \;\ge\; \Pr\left[\sum_{v\in U^*} w(v) \ge \frac{\varepsilon}{4}D^* \;\middle|\; A\right]\cdot \Pr[A].$$

We apply Lemma 4.1 to the set U^* and the sample S, with parameters m, ε, and $q = \frac{1}{2}$. The condition $\varepsilon = \sqrt{\frac{\ln n}{n}} < \frac{1-q}{q} = 1$ is satisfied for all natural numbers n. Since $m \le n$, this allows us to lower-bound the probability of event A by $cn^{-3/2}$. All the probabilities and expectations in the rest of the proof are conditioned on the event A.

Let us now consider the expected value of $\sum_{v\in U^*} w(v)$. Fix a particular demand pair $\{u_i, v_i\}$ that is separated by the optimal solution, and assume without loss of generality that $u_i \in U^*$ and $v_i \notin U^*$. Let p_u be the probability that $u_i \in S$, and p_v be the probability that $v_i \in S$. Then $p_u = \frac{|U^*\cap S|}{|U^*|} \ge (1+\varepsilon)/2$, $p_v = \frac{1}{2}$, and the two events are independent. So

$$\begin{aligned}
\Pr[u_i = s_i] = \Pr[u_i \in S \wedge v_i \notin S] \;&=\; p_u\cdot(1-p_v) \;\ge\; (1+\varepsilon)/4,\\
\Pr[u_i = t_i] = \Pr[u_i \notin S \wedge v_i \in S] \;&=\; (1-p_u)\cdot p_v \;\le\; (1-\varepsilon)/4.
\end{aligned}$$

Then the expected contribution of this demand pair to $\sum_{v\in U^*} w(v)$ is equal to

$$\Pr[u_i = s_i]\cdot d_i + \Pr[u_i = t_i]\cdot(-d_i) \;\ge\; d_i\cdot\frac{\varepsilon}{2}.$$

By linearity of expectation,

$$\mathrm{E}\left[\sum_{v\in U^*} w(v)\right] \;\geq\; D^*\cdot\frac{\varepsilon}{2}.$$

We now use Markov's inequality [17] to bound the desired probability. For this we define a nonnegative random variable $Y = D^* - \sum_{v\in U^*} w(v)$. Then $\mathrm{E}[Y] \leq (1-\varepsilon/2)D^*$. So

$$\begin{aligned}\Pr\left[\sum_{v\in U^*} w(v) \leq \frac{\varepsilon}{4}D^*\right] \;&=\; \Pr\left[Y \geq (1-\frac{\varepsilon}{4})D^*\right] \;\leq\; \frac{\mathrm{E}[Y]}{(1-\varepsilon/4)D^*}\\ &\leq\; \frac{1-\varepsilon/2}{1-\varepsilon/4} \;=\; 1-\frac{\varepsilon}{4-\varepsilon} \;\leq\; 1-\frac{\varepsilon}{4}\end{aligned}$$

It follows that

$$\Pr\left[\sum_{v\in U^*} w(v) \geq \frac{\varepsilon}{4}D^*\right] \;\geq\; \frac{\varepsilon}{4} \;=\; \frac{1}{4}\sqrt{\frac{\ln n}{n}} \;\geq\; \frac{1}{4\sqrt{n}},$$

concluding the proof of the lemma. □

Theorem 4.4. *For any feasible instance of SSC problem, Algorithm 1 returns a solution with cost at most $4\sqrt{\frac{n}{\ln n}}\cdot B$, with probability at least p.*

Proof. By Lemma 4.3, 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 $\frac{4n^2}{c}\ln(\frac{1}{1-p})$ iterations is at least p. Now, assuming that it does hold, the algorithm finds a set T such that

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

Applying Lemma 4.2, we get that $f(T)/\sum_{i:|T\cap\{u_i,v_i\}|=1} d_i < \alpha = 4\sqrt{\frac{n}{\ln n}}\cdot B$, which means that T is the required approximate solution. □

§ 5. Algorithms for 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 1. This allows us to obtain a bicriteria approximation for SBC in a similar way that Leighton and Rao [16] use their algorithm for sparsest cut on graphs to approximate balanced cut on graphs. Leighton and Rao present two versions of an algorithm for the balanced cut problem on graphs — one for undirected graphs, and one for directed graphs. The algorithm for undirected graphs has a better balance guarantee. We describe adaptations of these algorithms to the submodular version of the balanced cut problem. Our first algorithm extends the one for undirected graphs, and it works for symmetric submodular functions. For a given $b' \le 1/3$, it finds 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 $b' < b \le \frac{1}{2}$. The second algorithm works for arbitrary non-negative submodular functions and produces 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 $b' < b \le 1/2$.

§ 5.1. SBC algorithm for symmetric submodular functions

The algorithm for SBC on symmetric functions (Algorithm 2) repeatedly finds approximate weighted submodular sparsest cuts $(S_i, \bar{S}_i)$ and collects their smaller sides into the set T, until $(T, \bar{T})$ becomes b'-balanced. The algorithm and analysis basically follow Leighton and Rao [16], with the main difference being that instead of removing parts of the graph, we set the weights of the corresponding elements to zero. Then the obtained sets S_i are not necessarily disjoint.

Algorithm 2 Submodular balanced cut for symmetric functions. Input: V, f, w, $b' \le \frac{1}{3}$

```
1: Initialize $w' = w$, $i = 0$, $T = \emptyset$
2: while $w'(V) > (1 - b')w(V)$ do
3:     Let $S$ be a $\gamma$-approximate weighted SSC on $V$, $f$, and weights $w'$
4:     if $w'(S) \le w'(\bar{S})$ then let $S_i = S$ else let $S_i = \bar{S}$
5:     $w'(S_i) \leftarrow 0$;  $T \leftarrow T \cup S_i$;  $i \leftarrow i + 1$
6: end while
7: return $T$
```

Theorem 5.1. *If the system (V, f, w) contains a b-balanced cut of cost B, then Algorithm 2 finds a b'-balanced cut T with $f(T) = O\left(\frac{B}{b-b'}\sqrt{\frac{n}{\ln n}}\right)$, for a given $b' < b$, $b' \le \frac{1}{3}$.*

Proof. Algorithm 2 terminates in $O(n)$ iterations, as the weight of at least one new element is set to zero on line 5 (otherwise the cost of SSC solution would be infinite).

Now we consider $w(T)$. By the termination condition of the while loop, we know that when it exits, $w'(V) \le (1 - b')w(V)$, which means that w' has been set to zero for elements of total weight at least $b'w(V)$. But those are exactly the elements in T, so $w(T) \ge b'w(V)$. Now consider the last iteration of the loop. At the beginning of this iteration, we have $w'(V) > (1 - b')w(V)$, which means that at the end of it we have

$w'(V) > \frac{1}{2}(1-b')w(V)$, because the weight of the smaller (according to w') of S or $\bar{S}$ is set to zero. But $w'(V)$ at the end of the algorithm is exactly the weight of $\bar{T}$, which means that $w(\bar{T}) > \frac{1}{2}(1-b')w(V) \geq \frac{1}{3}w(V) \geq b'w(V)$, using the assumption $b' \leq 1/3$ twice. So the cut $(T, \bar{T})$ is b'-balanced.

Suppose that U^* is a b-balanced cut with $f(U^*) = B$. In any iteration i of the while loop, we know that two inequalities hold: $w'(U^*) + w'(\bar{U}^*) > (1-b')w(V)$ (by the loop condition), and $\max(w'(U^*), w'(\bar{U}^*)) \leq (1-b)w(V)$ (by b-balance). Given these inequalities, the minimum value that the product $w'(U^*) \cdot w'(\bar{U}^*)$ can have is $(b-b')w(V) \cdot (1-b)w(V)$. So with weights w', there is a solution to SSC with value

$$\frac{f(U^*)}{w'(U^*)w'(\bar{U}^*)} \leq \frac{B}{(b-b')w(V) \cdot (1-b)w(V)},$$

and the set S_i found by the γ-approximation algorithm satisfies

$$\frac{f(S_i)}{w'(S_i)w'(\bar{S}_i)} \leq \frac{\gamma B}{(b-b')w(V) \cdot (1-b)w(V)}.$$

Since in iteration i, $w'(S_i) = w(S_i \setminus \bigcup_{j=0}^{i-1} S_j)$, $w'(\bar{S}_i) \leq w(V)$, and $(1-b) \geq 1/2$,

$$f(S_i) \leq w(S_i \setminus \bigcup_{j=0}^{i-1} S_j) \frac{2B\gamma}{(b-b')w(V)}.$$

Now $f(T) \leq \sum_i f(S_i) \leq w(T) \cdot 2B\gamma/(b-b')w(V) = B \cdot O(\frac{\gamma}{b-b'})$. □

§ 5.2. SBC algorithm for general submodular functions

The algorithm for general functions (Algorithm 3) also repeatedly finds weighted submodular sparsest cuts $(S_i, \bar{S}_i)$, but it uses them to collect two sets: either it puts S_i into T_1, or it puts $\bar{S}_i$ into T_2. Thus, the values of $f(T_1)$ and $\bar{f}(T_2)$ can be bounded using the guarantee of the SSC algorithm (recall that $\bar{f}(S) = f(\bar{S})$).

Algorithm 3 Submodular balanced cut. Input: V, f, w, b'

```
1: Initialize $w' = w$, $i = 0$, $T_1 = T_2 = \emptyset$
2: while $w'(V) > (1-b')w(V)$ do
3:     Let $S_i$ be a $\gamma$-approximate weighted SSC on $V$, $f$, and weights $w'$
4:     if $w'(S_i) \leq w'(\bar{S}_i)$ then set $T_1 \leftarrow T_1 \cup S_i$; $w'(S_i) \leftarrow 0$; $i \leftarrow i+1$
5:     else set $T_2 \leftarrow T_2 \cup \bar{S}_i$; $w'(\bar{S}_i) \leftarrow 0$; $i \leftarrow i+1$
6: end while
7: if $w(T_1) \geq w(T_2)$ then return $T_1$ else return $\bar{T}_2$
```

Theorem 5.2. *If the system (V, f, w) contains a b-balanced cut of cost B, then Algorithm 3 finds a $b'/2$-balanced cut T with $f(T) = O\left(\frac{B}{b-b'}\sqrt{\frac{n}{\ln n}}\right)$, for a given $b' < b$.*

Proof. When the while loop exits, $w'(V) \leq (1-b')w(V)$, so the total weight of elements in T_1 and T_2 (the ones for which w' has been set to zero) is at least $b'w(V)$. So $\max(w(T_1), w(T_2)) \geq b'w(V)/2$. At the beginning of the last iteration of the loop, $w'(V) > (1-b')w(V)$. Since the weight of the smaller of S_i and $\bar{S}_i$ is set to zero, at the end of this iteration $w'(V) > \frac{1}{2}(1-b')w(V)$. Let T be the set output by the algorithm. Since $w'(T) = 0$, we have $w(\bar{T}) \geq w'(V) > \frac{1}{2}(1-b')w(V) \geq b'/2$, using $b' \leq 1/2$. Thus we have shown that Algorithm 3 outputs a $b'/2$-balanced cut.

The function values can be bounded as $f(T_1) = B \cdot O(\frac{\gamma}{b-b'})$ and $\bar{f}(T_2) = B \cdot O(\frac{\gamma}{b-b'})$ using a proof similar to that of Theorem 5.1. □

§ 6. Lower bound for approximating monotone submodular functions

We present a lower bound for the problem of approximating submodular functions everywhere, which holds even for the special case of monotone functions. 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:

$$f_1(S) \;=\; \min(|S|,\ \alpha), \qquad f_2(S) \;=\; \min\left(\beta + |S \cap \bar{R}|,\ |S|,\ \alpha\right). \tag{6.1}$$

Lemma 6.1. *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, it is easy to check (similarly to the proof of Lemma 3.1) that $\Pr[f_1(S) \neq f_2(S)]$ is maximized for sets S of size α. 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 α. Let us consider a different set, R', which is obtained by independently including each element of V with probability α/n. The expected size of R' is α, and the probability that $|R'| = \alpha$ is at least $1/(n+1)$. Then

$$\Pr[|S \cap R| > \beta] \;=\; \Pr[|S \cap R'| > \beta \mid |R'| = \alpha] \;\leq\; (n+1) \cdot \Pr[|S \cap R'| > \beta],$$

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

$$\Pr[|S \cap R'| > \beta] \;<\; \left(\frac{e^4}{5^5}\right)^{\frac{x^2}{25}} \;\leq\; 0.851^{x^2}.$$

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

Theorem 6.2. *Any algorithm that makes a polynomial number of oracle queries cannot approximate monotone submodular functions to a factor* $o\left(\sqrt{\frac{n}{\ln n}}\right)$.

Proof. Suppose that there is a γ-approximation algorithm for the problem, with $\gamma = o\left(\sqrt{\frac{n}{\ln n}}\right)$, which makes a polynomial number of oracle queries. Let $x = \sqrt{n}/2\gamma$, which satisfies $x^2 = \omega(\ln n)$. By Lemma 6.1, with high probability this algorithm produces the same output (say $\hat{f}$) if given as input either f_1 or f_2, defined in (6.1), with parameter x. Thus, by the algorithm's guarantee, $\hat{f}$ is simultaneously a γ-approximation for both f_1 and f_2. For the set R used in f_2, this guarantee implies that $f_1(R) \leq \gamma\hat{f}(R) \leq \gamma f_2(R)$. Since $f_1(R) = \alpha$ and $f_2(R) = \beta$, we have that $\gamma \geq \alpha/\beta = \sqrt{n}/x = 2\gamma$, which is a contradiction. □

§ 7. Approximating monotone two-partition submodular functions

Recall that a 2P function is one for which there is a set $R \subseteq V$ such that the value of $f(S)$ depends only on $|S \cap R|$ and $|S \cap \bar{R}|$. Our algorithm for approximating monotone 2P functions everywhere (Algorithm 4) uses the following observation.

Lemma 7.1. *Given two sets* S *and* T *such that* $|S| = |T|$, *but* $f(S) \neq f(T)$, *a 2P function can be found 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, $\{s_1, ..., s_k\}$, and an ordering of elements of T, $\{t_1, ..., t_k\}$, 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: $S_i = \{t_1, ..., t_i, s_{i+1}, ..., s_k\}$. 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 $U = \{t_1, ..., t_{i-1}, s_{i+1}, ..., s_k\}$, 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 obtained by querying sets W with all possible values of $|W \cap R|$ and $|W \cap \bar{R}|$. $\square$

Algorithm 4 Approximating monotone 2P function everywhere. Input: V, f, p

```
1: Query values of $f(\emptyset)$, $f(V)$, and $f(\{j\})$ for each $j \in V$
2: For each $i \in \{2, ..., n-1\}$, 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 find the function exactly, as described in Lemma 7.1.
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 7.2. *With probability at least p, the function $\hat{f}$ returned by Algorithm 4 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 7.1. 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 $\hat{R}$. 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 7.3. *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 $l = |S \cap \bar{R}|$. We observe that by monotonicity, $f(S) \ge f(k, 0)$ and $f(S) \ge f(0, l)$. Moreover, since $|S| = k + l \ge 2\sqrt{n}$, we have $\max(k, l) \ge \sqrt{n}$. So by Lemma 7.3, $f(S) \ge \frac{\max(k,l)}{2n} f(V) \ge \frac{f(V)}{2\sqrt{n}} = \hat{f}(S)$. □

To prove Lemma 7.3, we use several preliminary lemmas.

Definition 7.4. A pair of integers (k, l) with $k \le K$ and $l \le L$ is said to be *balanced* if it satisfies

$$l \cdot \frac{K}{L} - 2 \;\le\; k \;\le\; l \cdot \frac{K}{L} + 2. \tag{7.1}$$

Intuitively, a balanced pair (k, l) is one in which $\frac{k}{l}$ is close to $\frac{K}{L}$, so that a set S corresponding to this pair has the number of elements from R and $\bar{R}$ proportional to the sizes of the two sets.

Lemma 7.5. *Suppose that $m \le n$ elements are selected independently with probability $q \in [\frac{1}{n}, \frac{n-1}{n}]$ each, and let X denote the total number of selected elements. Then for any integer $x \in [0, m-1]$, $\;1/n^2 \;\le\; \Pr[X = x+1]/\Pr[X = x] \;\le\; n^2$.*

Proof.

$$\frac{\Pr[X = x+1]}{\Pr[X = x]} = \frac{\binom{m}{x+1} q^{x+1} (1-q)^{m-x-1}}{\binom{m}{x} q^x (1-q)^{m-x}} = \frac{m!\; x!\; (m-x)!\; q}{(x+1)!(m-x-1)!\; m!\; (1-q)}$$
$$= \frac{(m-x)\, q}{(x+1)(1-q)},$$

with the minimum value of $1/m(n-1) \ge 1/n^2$ achieved at $x = m-1$ and $q = \frac{1}{n}$, and the maximum value of $m(n-1) \le n^2$ achieved at $x = 0$ and $q = \frac{n-1}{n}$. □

Lemma 7.6. *If Algorithm 4 reaches step 4, then with probability at least p, for all balanced (k_1, l_1) and (k_2, l_2) such that $k_1 + l_1 = k_2 + l_2$, it holds that $f(k_1, l_1) = f(k_2, l_2)$. In other words, for all balanced pairs (k, l), the value of $f(k, l)$ depends only on $k + l$.*

Proof. The lemma follows if we show that with probability at least p, for each balanced (k, l) with $k + l < n$, the algorithm samples at least one set S corresponding to (k, l). This is because the algorithm verifies that the function value for the sets that it samples depends only on the set size.

So consider a specific balanced pair (k, l) and one random set S generated by the iteration $i = k + l$ of step 2 of the algorithm. The probability of sampling each element in this iteration is $q = \frac{i}{n} = \frac{k+l}{K+L}$. Using (7.1) and its equivalent $(k-2)L/K \le l \le (k+2)L/K$, we see that this probability satisfies the following:

$$\frac{k}{K} - \frac{2L}{Kn} \;\le\; q \;\le\; \frac{k}{K} + \frac{2L}{Kn} \quad \text{and} \quad \frac{l}{L} - \frac{2}{n} \;\le\; q \;\le\; \frac{l}{L} + \frac{2}{n}.$$

So the expected value of $|S \cap R|$ is $qK \in [k - 2L/n, k + 2L/n] \subseteq [k-2, k+2]$. Similarly, the expected value of $|S \cap \bar{R}|$ is $qL \in [l-2, l+2]$. Let μ_k be the most likely number of sampled elements when independently sampling K elements with probability q each. Then μ_k is equal to either $\lfloor qK \rfloor$ or $\lceil qK \rceil$. From above considerations and because k is an integer, we have that $\mu_k \in [k-2, k+2]$. Now, since μ_k is the most likely value, we know that $\Pr[|S \cap R| = \mu_k] \geq 1/(K+1) \geq 1/n$. By Lemma 7.5 (with $m = K$),

$$\Pr[|S \cap R| = k] \;\geq\; \Pr[|S \cap R| = \mu_k] \cdot n^{-2 \cdot |k - \mu_k|} \;\geq\; n^{-5}.$$

We similarly define μ_l, observe that $\mu_l \in [l-2, l+2]$, and conclude that $\Pr[|S \cap \bar{R}| = l] \geq n^{-5}$. Since the two events are independent, the probability that both of them occur, and thus that S corresponds to (k, l), is at least n^{-10}.

We observe that for any i, there are at most four balanced pairs (k, l) such that $k + l = i$. This is because if some pair (k, l) satisfies (7.1), then the pair $(k-4, l+4)$ doesn't satisfy it:

$$k - 4 \;\leq\; \left(l\frac{K}{L} + 2\right) - 4 \;=\; l\frac{K}{L} - 2 \;<\; (l+4)\frac{K}{L} - 2.$$

So there is a total of at most $4n$ pairs (k, l) for which we would like the algorithm to sample their corresponding sets. Since the number of trials for each value of $k + l$ is $n^{10} \ln\left(\frac{4n}{1-p}\right)$, the probability that a set corresponding to any particular pair (k, l) is *not* sampled is at most

$$\left(1 - n^{-10}\right)^{n^{10} \ln\left(\frac{4n}{1-p}\right)} \;\leq\; e^{-\ln\left(\frac{4n}{1-p}\right)} \;=\; \frac{1-p}{4n}.$$

Since there are at most $4n$ pairs of interest, by union bound we have that the probability that at least one of them remains unsampled is at most $(1-p)$. $\square$

Suppose the condition in Lemma 7.6 holds. Let us define a function $F(i)$ to be equal to $f(k, l)$ such that $k + l = i$ and (k, l) is balanced. $F(i)$ is defined for all $i \in \{0, ..., n\}$, since for any such i there is at least one balanced pair (k, l) with $k + l = i$.

Lemma 7.7. *$F(i)$ is a non-decreasing concave function.*

Proof. Let $\Delta(i) = F(i+1) - F(i)$. It suffices to show that the sequence of increments $\Delta(i)$ is non-negative and non-increasing. For any i, we define a pair $(k_i, l_i) = \left(\left\lfloor \frac{iK}{n} \right\rfloor, \left\lceil \frac{iL}{n} \right\rceil\right)$. It can be verified that all pairs (k_i, l_i) as well as $(k_i + 1, l_i)$ are balanced. Furthermore, $k_i + l_i = i$ (and consequently $k_i + 1 + l_i = i + 1$), so that $f(k_i + 1, l_i) - f(k_i, l_i) = \Delta(i)$. Also, both $\{k_i\}$ and $\{l_i\}$ are non-decreasing sequences. The decreasing marginal values of the submodular function f imply that $\Delta(i+1) = f(k_{i+1} + 1, l_{i+1}) - f(k_{i+1}, l_{i+1}) \leq f(k_i + 1, l_i) - f(k_i, l_i) = \Delta(i)$, showing that $\Delta(i)$'s are non-increasing. The monotonicity of f implies that they are also non-negative. $\square$

We next define two sequences of pairs, (k_i^K, l_i^K) and (k_i^L, l_i^L), ranging from $i = 0$ to $i = n$, which we call the K*-biased* sequence and the L*-biased* sequence, respectively. The properties of these two sequences will be used in the remainder of the proof. The definitions are inductive, with both sequences starting at $(0, 0)$.

$$(k_{i+1}^K,\ l_{i+1}^K) = \begin{cases} (k_i^K + 1,\ l_i^K) & \text{if } \ k_i^K \le l_i^K \cdot \frac{K}{L} \\ (k_i^K,\ l_i^K + 1) & \text{if } \ k_i^K > l_i^K \cdot \frac{K}{L} \end{cases}$$

$$(k_{i+1}^L,\ l_{i+1}^L) = \begin{cases} (k_i^L + 1,\ l_i^L) & \text{if } \ k_i^L < l_i^L \cdot \frac{K}{L} \\ (k_i^L,\ l_i^L + 1) & \text{if } \ k_i^L \ge l_i^L \cdot \frac{K}{L} \end{cases}$$

Let us call the change from (k_i, l_i) to (k_{i+1}, l_{i+1}) in either of the two sequences a K*-step* if the first component of the pair increases by one, and an L*-step* if the second component increases. The only difference between the two sequences is that when equality $k = l \cdot K/L$ holds, we take a K-step in the case of the K-biased sequence, and an L-step in the case of the L-biased sequence. For both sequences it holds that $k_i^K + l_i^K = k_i^L + l_i^L = i$, k_i^K and k_i^L range between 0 and K, and l_i^K and l_i^L range between 0 and L.

Lemma 7.8. *All pairs in the K-biased and L-biased sequences are balanced.*

Proof. The proof is by induction, and it is the same for both sequences, so we denote either sequence by (k_i, l_i). The first pair $(0, 0)$ is balanced. Now we assume that the pair (k_i, l_i) is balanced, and would like to show that the pair (k_{i+1}, l_{i+1}) is also balanced. Suppose $(k_{i+1}, l_{i+1}) = (k_i + 1, l_i)$. Then it must be that $k_i \le l_i \cdot \frac{K}{L}$. Then

$$l_i \cdot \frac{K}{L} - 2 \ \le\ k_i \ \le\ \ k_i + 1 \ \ \le\ l_i \cdot \frac{K}{L} + 1.$$

If $(k_{i+1}, l_{i+1}) = (k_i, l_i + 1)$, then it must be that $k_i \ge l_i \cdot \frac{K}{L}$. Then

$$(l_i + 1) \cdot \frac{K}{L} - 2 \ \le\ l_i \cdot \frac{K}{L} \ \le\ \ k_i \ \ \le\ l_i \cdot \frac{K}{L} + 2 \ \le\ (l_i + 1) \cdot \frac{K}{L} + 2,$$

with the leftmost inequality following because $K/L \le 1$. □

Lemma 7.9. *In the K-biased sequence, every K-step is followed by at most $\lceil \frac{L}{K} \rceil$ L-steps. In the L-biased sequence, every L-step is followed by at most one K-step.*

Proof. Suppose that the K-biased sequence, after some point (k, l), takes one K-step followed by $\lceil \frac{L}{K} \rceil$ L-steps, reaching the point $(k + 1, l + \lceil \frac{L}{K} \rceil)$. Since the step after (k, l) is a K-step, it must be that $k \le lK/L$. So

$$\left(l + \left\lceil \frac{L}{K} \right\rceil\right) \cdot \frac{K}{L} \ \ge\ l \cdot \frac{K}{L} + 1 \ \ge\ k + 1,$$

which means that the next step in the K-biased sequence will be a K-step.

Similarly, for the L-biased walk, suppose that from some point (k, l), the sequence takes an L-step, followed by a K-step, reaching the point $(k+1, l+1)$. Then $k \geq lK/L$ implies that

$$(l+1) \cdot \frac{K}{L} \;=\; l \cdot \frac{K}{L} + \frac{K}{L} \;\leq\; l \cdot \frac{K}{L} + 1 \;\leq\; k+1,$$

and thus the next step is an L-step. □

Proof of Lemma 7.3. To lower-bound the value of $f(k,0)$, we consider the K-biased walk from $(0,0)$ to a point (k,l') which is the last point before the K-step to $(k+1,\cdot)$. We let $f(k,0) = F(0) + \sum_{j=1}^{k} \delta(j)$, where $\delta(j) = f(j,0) - f(j-1,0)$. For each K-step in the K-biased walk, where $k_{i-1}^K = j-1$ and $k_i^K = j$, let $\Delta^K(j) = f(k_i^K, l_i^K) - f(k_{i-1}^K, l_{i-1}^K) = f(j, l_i^K) - f(j-1, l_{i-1}^K)$. By submodularity of f it follows that $\Delta^K(j) \leq \delta(j)$.

We claim that $\sum_{j=1}^{k} \Delta^K(j) \geq [f(k,l') - F(0)]/(1 + \lceil \frac{L}{K} \rceil)$. In other words, at least $1/(1 + \lceil \frac{L}{K} \rceil)$ fraction of the increase in $F(\cdot)$, as we proceed in the K-biased walk, is due to the K-steps. This follows from several observations. First, the K-biased walk starts with a K-step. Second, by Lemma 7.9, each K-step is followed by no more than $\lceil \frac{L}{K} \rceil$ L-steps. And third, $\Delta^K(j)$ is a decreasing sequence (by concavity of F).

Further, by concavity of F, we have that $f(k,l') \geq \frac{k+l'}{n} F(n)$. By definition of l', we have $l' \geq kL/K$. Also, $1 + \lceil \frac{L}{K} \rceil \leq 2(L/K+1)$. Putting everything together, we have

$$\begin{aligned} f(k,0) = F(0) + \sum_{j=1}^{k} \delta(j) \;&\geq\; F(0) + \sum_{j=1}^{k} \Delta^K(j) \;\geq\; F(0) + \frac{f(k,l') - F(0)}{1 + \lceil \frac{L}{K} \rceil} \;\geq\; \frac{f(k,l')}{1 + \lceil \frac{L}{K} \rceil} \\ &\geq \frac{k+l'}{n} \frac{F(n)}{2(L/K+1)} \;\geq\; \frac{k(L/K+1)}{n} \frac{F(n)}{2(L/K+1)} \;=\; \frac{k}{2n} F(n) \end{aligned}$$

To bound $f(0,l)$, we consider the L-biased walk from $(0,0)$ to (k',l) for some k'. Because of concavity of F, the L-steps in the walk account for at least half the increase in f, yielding $f(0,l) \geq \frac{1}{2} f(k',l)$. Also, $f(k',l) \geq \frac{k'+l}{n} F(n) \geq \frac{l}{n} F(n)$. So we get that $f(0,l) \geq \frac{l}{2n} F(n)$. □

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 FOCS*, pages 238–247, 2004.

[2] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proc. 36th STOC*, 2004.

[3] C. Chekuri, G. Calinescu, M. Pal, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *IPCO*, 2007.

[4] W.H. Cunningham. Minimum cuts, modular functions, and matroid polyhedra. *Networks*, 15:205–215, 1985.

[5] U. Feige, V. Mirrokni, and J. Vondrák. Maximizing non-monotone submodular functions. In *Proc. 48th FOCS*, 2007.

[6] M. Goemans, N. Harvey, S. Iwata, and V. Mirrokni. Approximating submodular functions everywhere. In *Proc. 20th SODA*, 2009.

[7] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[8] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.

[9] N. Harvey. *Matchings, Matroids and Submodular Functions*. PhD thesis, MIT, 2008.

[10] A. Hayrapetyan, C. Swamy, and É. Tardos. Network design for information networks. In *Proc. 16th SODA*, pages 933–942, 2005.

[11] B. Hoppe and É. Tardos. The quickest transshipment problem. *Math. Oper. Res.*, 25(1):36–62, 2000.

[12] S. Iwata. A faster scaling algorithm for minimizing submodular functions. *SIAM J. Comput.*, 32:833–840, 2003.

[13] S. Iwata. Submodular function minimization. *Math. Programming*, 112:45–64, 2008.

[14] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.

[15] J. Lee, V. Mirrokni, V. Nagarajan, and M. Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *Proc. 41th STOC*, 2009.

[16] F.T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46, 1999.

[17] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1990.

[18] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

[19] J. B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. In *IPCO*, 2007.

[20] M. Queyranne. Minimizing symmetric submodular functions. *Math. Programming*, 82:3–12, 1998.

[21] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. 40th STOC*, pages 255–263, 2008.

[22] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. of Combinatorial Theory, Ser. B*, 80(2):346–355, 2000.

[23] M. Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.

[24] Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. In *Proc. 49th FOCS*, 2008.

[25] Z. Svitkina and É. Tardos. Facility location with hierarchical facility costs. In *Proc. 17th SODA*, 2006.

[26] C. Swamy, Y. Sharma, and D. Williamson. Approximation algorithms for prize collecting steiner forest problems with submodular penalty functions. In *Proc. 18th SODA*, 2007.

[27] L.A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

[28] L. Zhao, H. Nagamochi, and T. Ibaraki. Greedy splitting algorithms for approximating multiway partition problems. *Mathematical Programming*, 102(1):167–183, 2005.