SIAM J. COMPUT. © 2015 Society for Industrial and Applied Mathematics
Vol. 44, No. 5, pp. 1384–1402

A TIGHT LINEAR TIME (1/2)-APPROXIMATION FOR UNCONSTRAINED SUBMODULAR MAXIMIZATION[*]

NIV BUCHBINDER[†], MORAN FELDMAN[‡], JOSEPH (SEFFI) NAOR[§], AND ROY SCHWARTZ[¶]

Abstract. We consider the Unconstrained Submodular Maximization problem in which we are given a nonnegative submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, and the objective is to find a subset $S \subseteq \mathcal{N}$ maximizing $f(S)$. This is one of the most basic submodular optimization problems, having a wide range of applications. Some well-known problems captured by Unconstrained Submodular Maximization include Max-Cut, Max-DiCut, and variants of Max-SAT and maximum facility location. We present a simple randomized linear time algorithm achieving a tight approximation guarantee of 1/2, thus matching the known hardness result of Feige, Mirrokni, and Vondrák [*SIAM J. Comput.*, 40 (2011), pp. 1133–1153]. Our algorithm is based on an adaptation of the greedy approach which exploits certain symmetry properties of the problem.

Key words. submodular functions, approximation algorithms, linear time

AMS subject classifications. 68R05, 68W20, 68W25

DOI. 10.1137/130929205

1. Introduction.

The study of combinatorial problems with submodular objective functions has recently attracted much attention and is motivated by the principle of economy of scale, prevalent in real-world applications. Submodular functions are also commonly used as utility functions in economics and algorithmic game theory. In combinatorial optimization, submodular functions and submodular maximization play a major role as several well-known examples of submodular functions include cuts in graphs and hypergraphs, rank functions of matroids, and covering functions. Given a set S and an element u, we denote the union $S \cup \{u\}$ by $S + u$ and the expression $S \setminus \{u\}$ by $S - u$. Given a ground set $\mathcal{N}$, a function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ is called *submodular* if for every $A \subseteq B \subseteq \mathcal{N}$ and $u \in \mathcal{N} \setminus B$, $f(A + u) - f(A) \geq f(B + u) - f(B)$.[1]

Perhaps the most basic submodular maximization problem is Unconstrained Submodular Maximization (USM). Given a nonnegative submodular fucntion f, the goal is to find a subset $S \subseteq \mathcal{N}$ maximizing $f(S)$. Note that there is no restriction on the choice of S, as any subset of $\mathcal{N}$ is a feasible solution. USM captures many well-studied problems such as Max-Cut, Max-DiCut [16, 21, 23, 27, 29, 38], variants of Max-SAT, and maximum facility location [1, 8, 9]. Moreover, USM has various applications in more practical settings, such as marketing in social networks [22], revenue maximization with discrete choice [2], and algorithmic game theory [11, 36].

[*]Received by the editors July 15, 2013; accepted for publication (in revised form) August 8, 2014; published electronically October 27, 2015. An earlier version of this paper appeared in *Proceedings of FOCS* 2012.
http://www.siam.org/journals/sicomp/44-5/92920.html

[†]Statistics and Operations Research Department, Tel Aviv University, Tel Aviv, Israel (niv.buchbinder@gmail.com). This author's work was supported by ISF grant 954/11 and BSF grant 2010426.

[‡]School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland (moran.feldman@epfl.ch).

[§]Computer Science Department, Technion, Haifa, Israel (naor@cs.technion.ac.il). This author's work was supported by ISF grant 954/11 and BSF grant 2010426.

[¶]Department of Computer Science, Princeton University, Princeton, NJ 08540 (roysch@cs.princeton.edu).

[1]Equivalently, for every $A, B \subseteq \mathcal{N}$: $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$.

USM has been studied since the sixties in the operations research community [2, 7, 17, 18, 19, 28, 31, 35]. Not surprisingly, since USM captures NP-hard problems, all these works provide algorithms that either solve special cases of the problem, provide exact algorithms whose time complexity cannot be efficiently bounded, or provide efficient algorithms whose output has no provable guarantee.

The first rigorous study of USM was conducted by Feige, Mirrokni, and Vondrák [12], who provided several constant approximation factor algorithms. They proved that a subset S chosen uniformly at random provides a (1/4)-approximation. Additionally, they also described two local search algorithms. The first uses f as the objective function and provides an approximation of 1/3. The second uses a noisy version of f as the objective function and achieves an improved approximation guarantee of 2/5. Gharan and Vondrák [15] showed that an extension of the last method, known as *simulated annealing*, can provide an improved approximation of roughly 0.41. Their algorithm, like the one in [12], uses local search with a noisy objective function. However, in [15] the noise decreases as the algorithm advances, as opposed to staying constant as in [12]. Feldman, Naor, and Schwartz [13] observed that if the simulated annealing algorithm of [15] outputs a relatively poor solution, then it must generate at some point a set S which is *structurally similar* to some optimal solution. Moreover, they showed that this structural similarity can be traded for value, providing an overall improved approximation of roughly 0.42.

It is important to note that for many special cases of USM, better approximation factors are known. For example, the seminal work of Goemans and Williamson [16] provides a 0.878-approximation for Max-Cut based on a semidefinite programming formulation, and Ageev and Sviridenko [1] provide an approximation of 0.828 for the maximum facility location problem.

On the negative side, Feige, Mirrokni, and Vondrák [12] studied the hardness of USM assuming the function f is given via a *value oracle*.[2] They proved that for any constant $\varepsilon > 0$, any algorithm achieving an approximation of $(1/2 + \varepsilon)$ requires an exponential number of oracle queries. This hardness result holds even if f is symmetric,[3] in which case an algorithm returning a random set containing every element with probability 1/2, independently, provides a matching (1/2)-approximation [12]. Recently, Dobzinski and Vondrák [10] proved that even if f has a compact representation (which is part of the input), the above hardness still holds assuming $RP \neq NP$.

1.1. Our results. In this paper we resolve the approximability of USM. We design a tight linear time randomized (1/2)-approximation for the problem. We begin by presenting a simple greedy-based deterministic algorithm that achieves a (1/3)-approximation for USM.

THEOREM 1.1. *There exists a deterministic linear time* (1/3)*-approximation algorithm for the Unconstrained Submodular Maximization problem.*

Then, we show that our analysis of the algorithm is tight, by providing for any arbitrarily small constant $\varepsilon > 0$ an instance for which the algorithm achieves only an approximation of $1/3 + \varepsilon$. To improve the performance beyond 1/3 we incorporate randomness into the choices of the algorithm. We obtain a randomized algorithm with an optimal approximation ratio for USM without increasing the time complexity.

THEOREM 1.2. *There exists a randomized linear time* (1/2)*-approximation algorithm for the Unconstrained Submodular Maximization problem.*

[2] The explicit representation of a submodular function might be exponential in the size of its ground set. Thus, it is standard practice to assume that the function is accessed via a value oracle. For a submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, the value oracle returns the value of $f(S)$ for a set $S \subseteq \mathcal{N}$.

[3] A submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ is symmetric if for every set $S \subseteq \mathcal{N}$, $f(S) = f(\mathcal{N} \setminus S)$.

In both Theorems 1.1 and 1.2 we assume that a single query to the value oracle takes $O(1)$ time. Thus, a linear time algorithm is an algorithm which makes $O(n)$ oracle queries and $O(n)$ other operations, where n is the size of the ground set $\mathcal{N}$.

Building on the above two theorems, we provide two additional approximation algorithms for Submodular Max-SAT and Submodular Welfare with two players (for the exact definition of these problems please refer to section 5). Both problems are already known to have tight approximation algorithms [14]. However, our algorithms run in linear time, thus significantly improving the time complexity while achieving the same performance guarantee.

THEOREM 1.3. *There exists a randomized linear time* $(3/4)$*-approximation algorithm for the* Submodular Max-SAT *problem.*

THEOREM 1.4. *There exists a randomized linear time* $(3/4)$*-approximation algorithm for the* Submodular Welfare *problem with two players.*

1.2. Techniques. The algorithms we present are based on a greedy approach. It is known that the straightforward greedy algorithm fails for USM. In contrast, Feldman, Naor, and Schwartz [14] recently showed that a continuous greedy algorithm does provide a $(1/e - o(1))$-approximation for maximizing any nonmonotone submodular function over a matroid. Recall, however, that better bounds than $1/e$ are already known for USM.

To understand the main ideas of our algorithm, consider a nonnegative submodular function f. Let us examine the complement of f, denoted by $\bar{f}$, defined as $\bar{f}(S) \triangleq f(\mathcal{N} \setminus S)$, for any $S \subseteq \mathcal{N}$. Note that since f is submodular, $\bar{f}$ is also submodular. Additionally, given an optimal solution $OPT \subseteq \mathcal{N}$ for USM with input f, $\mathcal{N} \setminus OPT$ is an optimal solution for $\bar{f}$, and both solutions have the exact same value. Any algorithm for USM has no way to distinguish whether it has oracle access to f or $\bar{f}$. Consider now the greedy algorithm. For f it starts from an empty solution and iteratively adds elements to it in a greedy fashion. Conversely, when applying the greedy algorithm to $\bar{f}$, one gets an algorithm for f that effectively starts with the solution $\mathcal{N}$ and then iteratively removes elements from it. Both algorithms are equally reasonable; however, both fail.

Despite the failure of the greedy algorithm when applied separately to either f or $\bar{f}$, we show that a correlated execution on both f and $\bar{f}$ provides a much better result. That is, we start with two solutions $\emptyset$ and $\mathcal{N}$. The algorithm considers the elements one at a time (in an arbitrary order). For each element it determines whether it should be added to the first solution or removed from the second one. Thus, after a single pass over the ground set $\mathcal{N}$, both solutions completely coincide and this is the algorithm's output. We show that a greedy choice in each step obtains an approximation guarantee of $1/3$, hence proving Theorem 1.1. To get Theorem 1.2, we use a natural randomized extension of the greedy rule, yielding an improved approximation guarantee of $1/2$.

1.3. Related work. Maximization problems of a nonnegative submodular function subject to various combinatorial constraints defining the feasible solutions have been studied extensively [14, 20, 32, 33, 40]. Similarly, minimization problems of a nonnegative submodular function subject to such constraints were studied as well [25, 26, 37]. Interestingly, the converse problem of unconstrained submodular minimization can be solved in polynomial time [34].

Another line of research deals with maximizing normalized *monotone* submodular functions,[4] again, subject to various combinatorial constraints. A continuous greedy

[4]A submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ is normalized if $f(\varnothing) = 0$ and monotone if for every two sets $A \subseteq B \subseteq \mathcal{N}$, $f(A) \leq f(B)$.

algorithm was given by Calinescu et al. [4] for maximizing a normalized monotone submodular function subject to a matroid constraint. Later, Lee, Sviridenko, and Vondrák [33] gave a local search algorithm achieving $1/p - \varepsilon$ approximation for maximizing such functions subject to the intersection of p matroids. Kulik, Shachnai, and Tamir [30] showed a $1-1/e-\varepsilon$ approximation algorithm for maximizing a normalized monotone submodular function subject to multiple knapsack constraints. In both the above mentioned works $\varepsilon > 0$ is an arbitrarily small constant. Recently, Chekuri, Vondrák, and Zenklusen [6] and Feldman, Naor, and Schwartz [14] gave nonmonotone counterparts of the continuous greedy algorithm of [4]. These results improve several nonmonotone submodular optimization problems. Some of the above results were generalized by Chekuri, Vondrák, and Zenklusen [5], who provide a dependent rounding technique for various polytopes, including matroid and matroid-intersection polytops. The advantage of this rounding technique is that it guarantees strong concentration bounds for submodular functions. Additionally, [6] defines a contention resolution rounding scheme which allows one to obtain approximations for combinations of constraint types.

2. A deterministic linear time (1/3)-approximation algorithm for USM.

In this section we present a deterministic linear time algorithm for USM. The algorithm proceeds in n iterations that correspond to some arbitrary order $u_1, \dots, u_n$ of the ground set $\mathcal{N}$. The algorithm maintains two solutions X and Y. Initially, we set the solutions to $X_0 = \emptyset$ and $Y_0 = \mathcal{N}$. In the ith iteration the algorithm either adds u_i to X_{i-1} or removes u_i from Y_{i-1}. This decision is done greedily based on the marginal gain of each of the two options. Eventually, after n iterations both solutions coincide, and we get $X_n = Y_n$; this is the output of the algorithm. A formal description of the algorithm appears as Algorithm 1.

Algorithm 1. Deterministic USM(f,$\mathcal{N}$).

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

The rest of this section is devoted to proving Theorem 1.1, i.e., we prove that the approximation ratio of Algorithm 1 is 1/3. In section 2.1 we show that our analysis of Algorithm 1 is tight. Let us begin with the following useful lemma.

Lemma 2.1. *For every $1 \leq i \leq n$, $a_i + b_i \geq 0$.*

Proof. Notice that $(X_{i-1}+u_i) \cup (Y_{i-1}-u_i) = Y_{i-1}$ and $(X_{i-1}+u_i) \cap (Y_{i-1}-u_i) = X_{i-1}$. By combining both observations with submodularity, one gets

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

Define $OPT_i \triangleq (OPT \cup X_i) \cap Y_i$, i.e., OPT_i agrees with X_i (and Y_i) on the first i elements and with OPT on the last $n-i$ elements. Thus, $OPT_0 = OPT$, and the output of the algorithm is $OPT_n = X_n = Y_n$. Examine the sequence $f(OPT_0), \dots, f(OPT_n)$, which starts with $f(OPT)$ and ends with the value of the output of the algorithm. The main idea of the proof is to bound the total loss of value along this sequence. This goal is achieved by the following lemma, which upper bounds the loss in value between every two consecutive elements of the sequence. Formally, the lemma shows that the loss of value, i.e., $f(OPT_{i-1}) - f(OPT_i)$, is no more than the *total* increase in value of both solutions maintained by the algorithm, i.e., $[f(X_i) - f(X_{i-1})] + [f(Y_i) - f(Y_{i-1})]$.

LEMMA 2.2. *For every* $1 \leq i \leq n$*,* $f(OPT_{i-1}) - f(OPT_i) \leq [f(X_i) - f(X_{i-1})] + [f(Y_i) - f(Y_{i-1})]$.

Before proving Lemma 2.2, let us show that Theorem 1.1 follows from it.

Proof of Theorem 1.1. Summing up Lemma 2.2 for every $1 \leq i \leq n$ gives

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

The above sum is telescopic. Collapsing it, we get

$$f(OPT_0) - f(OPT_n) \leq [f(X_n) - f(X_0)] + [f(Y_n) - f(Y_0)] \leq f(X_n) + f(Y_n).$$

Recalling the definitions of OPT_0 and OPT_n, we obtain that $f(X_n) = f(Y_n) \geq f(OPT)/3$. ∎

It is now left to prove Lemma 2.2.

Proof of Lemma 2.2. Assume without loss of generality that $a_i \geq b_i$, i.e., $X_i \leftarrow X_{i-1} + u_i$ and $Y_i \leftarrow Y_{i-1}$ (the other case is analogous). Notice that in this case $OPT_i = (OPT \cup X_i) \cap Y_i = OPT_{i-1} + u_i$ and $Y_i = Y_{i-1}$. Thus, the inequality that we need to prove becomes

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

We now consider two cases. If $u_i \in OPT$, then the left-hand side of the last inequality is 0, and all we need to show is that a_i is nonnegative. This is true since $a_i + b_i \geq 0$ by Lemma 2.1, and we assumed $a_i \geq b_i$.

If $u_i \notin OPT$, then also $u_i \notin OPT_{i-1}$, and thus

$$f(OPT_{i-1}) - f(OPT_{i-1} + u_i) \leq f(Y_{i-1} - u_i) - f(Y_{i-1}) = b_i \leq a_i.$$

The first inequality follows by submodularity: $OPT_{i-1} = ((OPT \cup X_{i-1}) \cap Y_{i-1}) \subseteq Y_{i-1} - u_i$ (recall that $u_i \in Y_{i-1}$ and $u_i \notin OPT_{i-1}$). The second inequality follows from our assumption that $a_i \geq b_i$. ∎

2.1. Tight example. In this section we show that the analysis of Algorithm 1 is tight.

THEOREM 2.3. *For an arbitrarily small constant* $\varepsilon > 0$*, there exists a submodular function for which Algorithm* 1 *provides only* $(1/3 + \varepsilon)$*-approximation.*

Proof. The proof is done by analyzing Algorithm 1 on the cut function of the weighted digraph given in Figure 1. The maximum weight cut in this digraph is $\{u_1, u_4, u_5\}$. This cut has weight of $6 - 2\varepsilon$. We claim that Algorithm 1 outputs the cut $\{u_2, u_3, u_4, u_5\}$, whose value is only 2 (assuming the nodes of the graph are considered at a given order). Hence, the approximation guarantee of Algorithm 1 on the above instance is at most

$$\frac{2}{6 - 2\varepsilon} = \frac{1}{3 - \varepsilon} \leq \frac{1}{3} + \varepsilon.$$

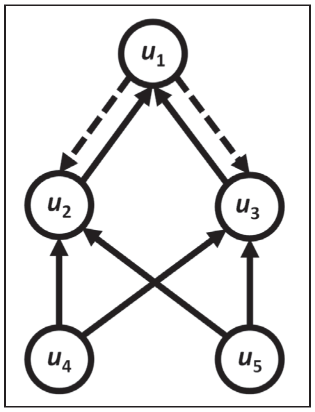

FIG. 1. *Tight example for Algorithm* 1. *The weight of the dashed edges is* $1-\varepsilon$. *All other edges have a weight of* 1.

Let us track the execution of the algorithm. Let X_i, Y_i be the solutions maintained by the algorithm. Initially $X_0 = \emptyset, Y_0 = \{u_1, u_2, u_3, u_4, u_5\}$. Note that in case of a tie ($a_i = b_i$), Algorithm 1 takes the node u_i.

1. In the first iteration the algorithm considers u_1. Adding this node to X_0 increases the value of this solution by $2 - 2\varepsilon$. On the other hand, removing this node from Y_0 increases the value of Y_0 by 2. Hence, $X_1 \leftarrow X_0, Y_1 \leftarrow Y_0 - u_1$.

2. Let us inspect the next two iterations in which the algorithm considers u_2, u_3. One can easily verify that these two iterations are independent, hence, we consider only u_2. Adding u_2 to X_1 increases its value by 1. On the other hand, removing u_2 from $Y_1 = \{u_2, u_3, u_4, u_5\}$ also increases the value of Y_1 by 1. Thus, the algorithm adds u_2 to X_1. Since u_2 and u_3 are symmetric, the algorithm also adds u_3 to X_1. Thus, at the end of these two iterations $X_3 = X_1 \cup \{u_2, u_3\} = \{u_2, u_3\}, Y_3 = \{u_2, u_3, u_4, u_5\}$.

3. Finally, the algorithm considers u_4 and u_5. These two iterations are also independent so we consider only u_4. Adding u_4 to X_3 does not increase the value of X_3. Also removing u_4 from Y_3 does not increase the value of Y_3. Thus, the algorithm adds u_4 to X_3. Since u_4 and u_5 are symmetric, the algorithm also adds u_5 to X_3. Thus, we get $X_5 = Y_5 = \{u_2, u_3, u_4, u_5\}$. $\square$

3. A randomized linear time (1/2)-approximation algorithm for USM.

In this section we present a randomized algorithm achieving a tight (1/2)-approximation for USM and thus prove Theorem 1.2. Algorithm 1 (presented in section 2) compared the marginal profits a_i and b_i. Based on this comparison the algorithm made a greedy deterministic decision whether to include or exclude u_i from its output. The random algorithm we next present makes a "smoother" decision, based on the values a_i and b_i. In each step, if both a_i and b_i are nonnegative, it randomly chooses whether to include or exclude u_i with probability proportional to the ratio between a_i and b_i. A formal description of the algorithm appears as Algorithm 2.

The rest of this section is devoted to proving that Algorithm 2 provides an approximation ratio of 1/2 for USM. Let us begin the analysis of Algorithm 2 with the introduction of some notation. Notice that for every $1 \leq i \leq n$, X_i and Y_i are now random variables denoting the sets of elements in the two solutions generated by the algorithm at the end of the ith iteration. As in section 2, let us define the following random variable: $OPT_i \triangleq (OPT \cup X_i) \cap Y_i$. Note that, again, $X_0 = \emptyset$, $Y_0 = \mathcal{N}$, and

Algorithm 2. Randomized $\mathsf{USM}(f, \mathcal{N})$.

```
1: $X_0 \leftarrow \emptyset$, $Y_0 \leftarrow \mathcal{N}$.
2: for $i = 1$ to $n$ do
3:     $a_i \leftarrow f(X_{i-1} + u_i) - f(X_{i-1})$.
4:     $b_i \leftarrow f(Y_{i-1} - u_i) - f(Y_{i-1})$.
5:     $a'_i \leftarrow \max\{a_i, 0\}$, $b'_i \leftarrow \max\{b_i, 0\}$.
6:     with probability $a'_i/(a'_i + b'_i)^*$ do: $X_i \leftarrow X_{i-1} + u_i$, $Y_i \leftarrow Y_{i-1}$.
7:     else (with the complement probability $b'_i/(a'_i + b'_i)$) do: $X_i \leftarrow X_{i-1}$,
       $Y_i \leftarrow Y_{i-1} - u_i$.
8: return $X_n$ (or equivalently $Y_n$).
```

* If $a'_i = b'_i = 0$, we assume $a'_i/(a'_i + b'_i) = 1$.

$OPT_0 = OPT$. Additionally, the following always holds: $OPT_n = X_n = Y_n$. The analysis of the approximation ratio is similar to that of the deterministic algorithm in section 2. We consider the sequence $\mathbb{E}[f(OPT_0)], \dots, \mathbb{E}[f(OPT_n)]$. This sequence starts with $f(OPT)$ and ends with the expected value of the algorithm's output. The following lemma upper bounds the loss between every two consecutive elements in the sequence. Formally, $\mathbb{E}[f(OPT_{i-1}) - f(OPT_i)]$ is upper bounded by the *average* expected change in the value of the two solutions maintained by the algorithm, i.e., $1/2 \cdot \mathbb{E}\left[f(X_i) - f(X_{i-1}) + f(Y_i) - f(Y_{i-1})\right]$.

Lemma 3.1. *For every $1 \leq i \leq n$,*

$$\mathbb{E}[f(OPT_{i-1}) - f(OPT_i)] \leq \frac{1}{2} \cdot \mathbb{E}\left[(f(X_i) - f(X_{i-1}) + f(Y_i) - f(Y_{i-1})\right], \tag{3.1}$$

where expectations are taken over the random choices of the algorithm.

Before proving Lemma 3.1, let us show that Theorem 1.2 follows from it.

Proof of Theorem 1.2. Summing up Lemma 3.1 for every $1 \leq i \leq n$ gives

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

The above sum is telescopic. Collapsing it, we get

$$\begin{aligned}\mathbb{E}[f(OPT_0) - f(OPT_n)] &\leq \frac{1}{2} \cdot \mathbb{E}[f(X_n) - f(X_0) + f(Y_n) - f(Y_0)] \\ &\leq \frac{\mathbb{E}[f(X_n) + f(Y_n)]}{2}.\end{aligned}$$

Recalling the definitions of OPT_0 and OPT_n, we obtain $\mathbb{E}[f(X_n)] = \mathbb{E}[f(Y_n)] \geq f(OPT)/2$. ☐

It is left to prove Lemma 3.1.

Proof of Lemma 3.1. Notice that it suffices to prove inequality (3.1) conditioned on any event of the form $X_{i-1} = S_{i-1}$, where $S_{i-1} \subseteq \{u_1, \dots, u_{i-1}\}$, for which the probability that $X_{i-1} = S_{i-1}$ is nonzero. Hence, fix such an event corresponding to a set S_{i-1}. The rest of the proof implicitly assumes everything is conditioned on this event. Notice that due to the conditioning, the following random variables become constants:

1. $Y_{i-1} = S_{i-1} \cup \{u_i, \ldots, u_n\}$.
2. $OPT_{i-1} \triangleq (OPT \cup X_{i-1}) \cap Y_{i-1} = S_{i-1} \cup (OPT \cap \{u_i, \ldots, u_n\})$.
3. a_i and b_i.

Moreover, by Lemma 2.1, $a_i + b_i \geq 0$. Thus, it cannot be the case that both a_i, b_i are strictly less than zero. Hence, we only need to consider the following three cases:

Case 1 *(*$\mathbf{a_i \geq 0}$ *and* $\mathbf{b_i \leq 0}$*)*. In this case $a'_i/(a'_i + b'_i) = 1$, and so the following always happen: $Y_i = Y_{i-1} = S_{i-1} \cup \{u_i, \ldots, u_n\}$ and $X_i \leftarrow S_{i-1} + u_i$. Hence, $f(Y_i) - f(Y_{i-1}) = 0$. Also, by our definition $OPT_i = (OPT \cup X_i) \cap Y_i = OPT_{i-1} + u_i$. Thus, we are left to prove that

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

If $u_i \in OPT$, then the left-hand side of the last inequality is 0, which is clearly not larger than the nonnegative $a_i/2$. If $u_i \notin OPT$, then

$$f(OPT_{i-1}) - f(OPT_{i-1} + u_i) \leq f(Y_{i-1} - u_i) - f(Y_{i-1}) = b_i \leq 0 \leq a_i/2.$$

The first inequality follows from submodularity since $OPT_{i-1} \triangleq (OPT \cup X_{i-1}) \cap Y_{i-1} \subseteq Y_{i-1} - u_i$ (note that $u_i \in Y_{i-1}$ and $u_i \notin OPT_{i-1}$).

Case 2 *(*$\mathbf{a_i < 0}$ *and* $\mathbf{b_i \geq 0}$*)*. This case is analogous to the previous one, and therefore we omit its proof.

Case 3 *(*$\mathbf{a_i \geq 0}$ *and* $\mathbf{b_i > 0}$*)*. In this case $a'_i = a_i, b'_i = b_i$. Therefore, with probability $a_i/(a_i + b_i)$ the following events happen: $X_i \leftarrow X_{i-1} + u_i$ and $Y_i \leftarrow Y_{i-1}$; while with probability $b_i/(a_i + b_i)$ the following events happen: $X_i \leftarrow X_{i-1}$ and $Y_i \leftarrow Y_{i-1} - u_i$. Thus,

(3.2)

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

Next, we upper bound $\mathbb{E}[f(OPT_{i-1}) - f(OPT_i)]$. As $OPT_i = (OPT \cup X_i) \cap Y_i$, we get

(3.3)

$$\begin{aligned}
\mathbb{E}[f(OPT_{i-1}) - f(OPT_i)] &= \frac{a_i}{a_i + b_i} \cdot [f(OPT_{i-1}) - f(OPT_{i-1} + u_i)] \\
&\quad + \frac{b_i}{a_i + b_i} \cdot [f(OPT_{i-1}) - f(OPT_{i-1} - u_i)] \leq \frac{a_i b_i}{a_i + b_i}.
\end{aligned}$$

The final inequality follows by considering two cases. Note first that $u_i \in Y_{i-1}$ and $u_i \notin X_{i-1}$. If $u_i \notin OPT_{i-1}$, then the second term of the left-hand side of the last inequality equals zero. Moreover, $OPT_{i-1} \triangleq (OPT \cup X_{i-1}) \cap Y_{i-1} \subseteq Y_{i-1} - u_i$, and therefore, by submodularity,

$$f(OPT_{i-1}) - f(OPT_{i-1} + u_i) \leq f(Y_{i-1} - u_i) - f(Y_{i-1}) = b_i.$$

If $u_i \in OPT_{i-1}$, then the first term of the left-hand side of inequality (3.3) equals zero, and we also have $X_{i-1} \subseteq ((OPT \cup X_{i-1}) \cap Y_{i-1}) - u_i = OPT_{i-1} - u_i$. Thus, by submodularity,

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

By (3.2) and (3.3), inequality (3.1) holds if

$$\frac{a_i b_i}{a_i + b_i} \leq \frac{1}{2} \cdot \left(\frac{a_i^2 + b_i^2}{a_i + b_i} \right),$$

which can easily be verified. □

4. A tight (1/2)-approximation for USM using fractional values. In section 3 we presented Algorithm 2, a randomized optimal algorithm for USM. In this section we present Algorithm 3, the continuous counterpart of Algorithm 2. This algorithm achieves the same approximation ratio (up to low order terms) but maintains a fractional inner state.

In order to describe Algorithm 3, we need some notation. Given ground set $\mathcal{N}$ and a vector $\vec{x} \in [0,1]^{\mathcal{N}}$, the random subset $\mathtt{R}(\vec{x}) \subseteq \mathcal{N}$ contains each element $u \in \mathcal{N}$ independently with probability x_u. Given a function $f : 2^{\mathcal{N}} \to \mathbb{R}$, its multilinear extension is a function $F : [0,1]^{\mathcal{N}} \to \mathbb{R}$, whose value at a vector $\vec{x} \in [0,1]^{\mathcal{N}}$ is the expected value of f over $\mathtt{R}(\vec{x})$. Formally, for every $\vec{x} \in [0,1]^{\mathcal{N}}$, $F(\vec{x}) \triangleq \mathbb{E}[\mathtt{R}(\vec{x})] = \sum_{S \subseteq \mathcal{N}} f(S) \prod_{u \in S} x_u \prod_{u \notin S} (1 - x_u)$. For two vectors $\vec{x}, \vec{y} \in [0,1]^{\mathcal{N}}$, we use $\vec{x} \vee \vec{y}$ and $\vec{x} \wedge \vec{y}$ to denote the coordinatewise maximum and minimum, respectively, of $\vec{x}$ and $\vec{y}$ (formally, $(\vec{x} \vee \vec{y})_u = \max\{x_u, y_u\}$ and $(\vec{x} \wedge \vec{y})_u = \min\{x_u, y_u\}$).

We abuse notation both in the description of the algorithm and in its analysis, and we unify a set with its characteristic vector. We also assume we have an oracle access to the multilinear extension F. If this is not the case, then the value of F can be approximated arbitrarily well using sampling.

ALGORITHM 3. MULTILINEAR USM$(f, \mathcal{N})$.

```
1: $\vec{x}_0 \leftarrow \emptyset$, $\vec{y}_0 \leftarrow \mathcal{N}$.
2: for $i = 1$ to $n$ do
3:     $a'_i \leftarrow F(\vec{x}_{i-1} + \{u_i\}) - F(\vec{x}_{i-1})$.
4:     $b'_i \leftarrow F(\vec{y}_{i-1} - \{u_i\}) - F(\vec{y}_{i-1})$.
5:     $a'_i \leftarrow \max\{a_i, 0\}$, $b'_i \leftarrow \max\{b_i, 0\}$.
6:     $\vec{x}_i \leftarrow \vec{x}_{i-1} + \frac{a'_i}{a'_i + b'_i} \cdot \{u_i\}^*$.
7:     $\vec{y}_i \leftarrow \vec{y}_{i-1} - \frac{b'_i}{a'_i + b'_i} \cdot \{u_i\}^*$.
8: return a random set $\mathtt{R}(x_n)$ (or equivalently $\mathtt{R}(y_n)$).
```

* If $a'_i = b'_i = 0$, we assume $a'_i/(a'_i + b'_i) = 1$ and $b'_i/(a'_i + b'_i) = 0$.

The main difference between Algorithms 2 and 3 is that Algorithm 2 chooses each element with some probability, whereas Algorithm 3 assigns a fractional value to the element. This requires the following modifications to the algorithm:

(i) The sets $X_i, Y_i \subseteq 2^{\mathcal{N}}$ are replaced by vectors $\vec{x}_i, \vec{y}_i \in [0,1]^{\mathcal{N}}$.

(ii) Algorithm 3 uses the multilinear extension F instead of the original submodular function f.

THEOREM 4.1. *Assuming an oracle access to F, Algorithm 3 is a $1/2$-approximation algorithm for USM.*

The rest of this section is devoted to proving Theorem 4.1. We begin with the following technical lemma, whose proof is omitted since it is almost identical to the proof of Lemma 2.1.

LEMMA 4.2. *For every $1 \leq i \leq n$, $a_i + b_i \geq 0$.*

Similarly to section 3, define $\vec{o}_i \triangleq (OPT \vee \vec{x}_i) \wedge \vec{y}_i$, and examine the sequence of values $F(\vec{o}_0), \dots, F(\vec{o}_n)$. Notice that $\vec{o}_0 = OPT$, i.e., the sequence starts with the value of an optimal solution, and that $\vec{o}_n = x_n = y_n$, i.e., the sequence ends at a fractional point whose value is the expected value of the algorithm's output. The following lemma upper bounds the loss between every two consecutive elements in the sequence. Formally, $F(\vec{o}_{i-1}) - F(\vec{o}_i)$ is upper bounded by the *average* change in the value of the two solutions maintained by the algorithm, i.e., $1/2 \cdot [F(\vec{x}_i) - F(\vec{x}_{i-1}) + F(\vec{y}_i) - F(\vec{y}_{i-1})]$.

LEMMA 4.3. *For every $1 \leq i \leq n$, $F(\vec{o}_{i-1}) - F(\vec{o}_i) \leq \frac{1}{2} \cdot [F(\vec{x}_i) - F(\vec{x}_{i-1}) + F(\vec{y}_i) - F(\vec{y}_{i-1})]$.*

Before proving Lemma 4.3, let us show that Theorem 4.1 follows from it.

Proof of Theorem 4.1. Summing up Lemma 4.3 for every $1 \leq i \leq n$ gives

$$\sum_{i=1}^{n}[F(\vec{o}_{i-1}) - F(\vec{o}_i)] \leq \frac{1}{2} \cdot \sum_{i=1}^{n}[F(\vec{x}_i) - F(\vec{x}_{i-1})] + \frac{1}{2} \cdot \sum_{i=1}^{n}[F(\vec{y}_i) - F(\vec{y}_{i-1})].$$

The above sum is telescopic. Collapsing it, we get

$$F(\vec{o}_0) - F(\vec{o}_n) \leq \frac{1}{2} \cdot [F(\vec{x}_n) - F(\vec{x}_0)] + \frac{1}{2} \cdot [F(\vec{y}_n) - F(\vec{y}_0)] \leq \frac{F(\vec{x}_n) + F(\vec{y}_n)}{2}.$$

Recalling the definitions of $\vec{o}_0$ and $\vec{o}_n$, we obtain that $F(\vec{x}_n) = F(\vec{y}_n) \geq f(OPT)/2$. ◻

It is left to prove Lemma 4.3.

Proof of Lemma 4.3. By Lemma 4.2, $a_i + b_i \geq 0$; therefore, it cannot be that both a_i, b_i are strictly less than zero. Thus, we have three cases to consider.

Case 1 *($\mathbf{a_i \geq 0}$ and $\mathbf{b_i \leq 0}$).* In this case $a'_i/(a'_i + b'_i) = 1$, and so $\vec{y}_i \leftarrow \vec{y}_{i-1}$, $\vec{x}_i \leftarrow \vec{x}_{i-1} \vee \{u_i\}$. Hence, $F(\vec{y}_i) - F(\vec{y}_{i-1}) = 0$. Also, by our definition $\vec{o}_i = (OPT \vee \vec{x}_i) \wedge \vec{y}_i = \vec{o}_{i-1} \vee \{u_i\}$. Thus, we are left to prove that

$$F(\vec{o}_{i-1}) - F(\vec{o}_{i-1} \vee \{u_i\}) \leq \frac{1}{2} \cdot [F(\vec{x}_i) - F(\vec{x}_{i-1})] = a_i/2.$$

If $u_i \in OPT$, then the left-hand side of the above equation is 0, which is clearly no larger than the nonnegative $a_i/2$. If $u_i \notin OPT$, then

$$F(\vec{o}_{i-1}) - F(\vec{o}_{i-1} \vee \{u_i\}) \leq F(\vec{y}_{i-1} - \{u_i\}) - F(\vec{y}_{i-1}) = b_i \leq 0 \leq a_i/2.$$

The first inequality follows from submodularity since $\vec{o}_{i-1} = ((OPT \vee \vec{x}_{i-1}) \wedge \vec{y}_{i-1}) \leq \vec{y}_{i-1} - \{u_i\}$ (note that in this case $(\vec{y}_{i-1})_{u_i} = 1$ and $(\vec{o}_{i-1})_{u_i} = 0$).

Case 2 *($\mathbf{a_i < 0}$ and $\mathbf{b_i \geq 0}$).* This case is analogous to the previous one, and therefore we omit its proof.

Case 3 ($\mathbf{a_i} \geq \mathbf{0}$ *and* $\mathbf{b_i} > \mathbf{0}$)*.* In this case $a'_i = a_i, b'_i = b_i$ and so, $\vec{x}_i \leftarrow \vec{x}_{i-1} + \frac{a_i}{a_i+b_i} \cdot \{u_i\}$ and $\vec{y}_i \leftarrow \vec{y}_{i-1} - \frac{b_i}{a_i+b_i} \cdot \{u_i\}$. Therefore, we have

$$\text{(4.1)}\quad \begin{aligned} F(\vec{x}_i) - F(\vec{x}_{i-1}) &= \left[\frac{a_i}{a_i+b_i} \cdot F(\vec{x}_{i-1} \vee \{u_i\}) + \frac{b_i}{a_i+b_i} \cdot F(\vec{x}_{i-1})\right] - F(\vec{x}_{i-1}) \\ &= \frac{a_i}{a_i+b_i} \cdot [F(\vec{x}_{i-1} \vee \{u_i\}) - F(\vec{x}_{i-1})] = \frac{a_i^2}{a_i+b_i}. \end{aligned}$$

A similar argument shows that

$$\text{(4.2)}\qquad F(\vec{y}_i) - F(\vec{y}_{i-1}) = \frac{b_i^2}{a_i+b_i}.$$

Next, we upper bound $F(\vec{o}_{i-1}) - F(\vec{o}_i)$. For simplicity, let us assume $u_i \notin OPT$ (the proof for the other case is similar). Recall that $\vec{o}_i = (OPT \vee \vec{x}_i) \wedge \vec{y}_i$.

$$\text{(4.3)}\quad \begin{aligned} F(\vec{o}_{i-1}) - F(\vec{o}_i) &= F(\vec{o}_{i-1}) - F\left(\vec{o}_{i-1} \vee \frac{a_i}{a_i+b_i} \cdot \{u_i\}\right) \\ &= F(\vec{o}_{i-1}) - \left[\frac{a_i}{a_i+b_i} \cdot F(\vec{o}_{i-1} \vee \{u_i\}) + \frac{b_i}{a_i+b_i} \cdot F(\vec{o}_{i-1})\right] \\ &= \frac{a_i}{a_i+b_i} \cdot [F(\vec{o}_{i-1}) - F(\vec{o}_{i-1} \vee \{u_i\})] \\ &\leq \frac{a_i}{a_i+b_i} \cdot [F(\vec{y}_{i-1} - \{u_i\}) - F(\vec{y}_{i-1})] = \frac{a_i b_i}{a_i+b_i}. \end{aligned}$$

The inequality follows from the submodularity of f since $\vec{o}_{i-1} = ((OPT \vee \vec{x}_{i-1}) \wedge \vec{y}_{i-1}) \leq \vec{y}_{i-1} - \{u_i\}$ (note again that in this case $(\vec{y}_{i-1})_{u_i} = 1$ and $(\vec{o}_{i-1})_{u_i} = 0$). By (4.1), (4.2), and (4.3), the lemma holds if

$$\frac{a_i b_i}{a_i+b_i} \leq \frac{1}{2} \cdot \frac{a_i^2 + b_i^2}{a_i+b_i},$$

which can easily be verified. ∎

5. Linear time approximations for Submodular Max-SAT and Submodular Welfare with two players.

In this section we build upon ideas from the previous sections to obtain linear time tight (3/4)-approximation algorithms for two problems: Submodular Max-SAT (SSAT) and Submodular Welfare (SW) with two players (both problems are defined below). In contrast to USM, tight approximations are already known for the above two problems [14]. However, the algorithms we present in this work, in addition to providing tight approximations, also run in linear time.

5.1. A linear time tight (3/4)-approximation for SSAT.

Recall that a submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}$ is normalized if $f(\varnothing) = 0$ and monotone if for every two sets $A \subseteq B \subseteq \mathcal{N}$, $f(A) \leq f(B)$. In SSAT we are given a conjunctive normal form (CNF) formula Ψ with a set $\mathcal{C}$ of clauses over a set $\mathcal{N}$ of variables, and a normalized monotone submodular function $f : 2^{\mathcal{C}} \to \mathbb{R}^+$ over the set of clauses. Given an assignment $\phi : \mathcal{N} \to \{0, 1\}$, let $C(\phi) \subseteq \mathcal{C}$ be the set of clauses satisfied by ϕ. The goal is to find an assignment ϕ maximizing $f(C(\phi))$.

Usually, an assignment ϕ can give each variable exactly a single truth value. However, for the sake of the algorithm we extend the notion of assignments and think of an extended assignment ϕ' which is a relation $\phi' \subseteq \mathcal{N} \times \{0, 1\}$. That is, the assignment ϕ' can assign up to two truth values to each variable. A clause C is satisfied by an (extended) assignment ϕ' if there exists a positive literal in the clause which is assigned the truth value 1, or there exists a negative literal in the clause which is assigned the truth value 0. Note again that it might happen that some variables are assigned both 0 and 1. Note also that an assignment is a feasible solution to the original problem if and only if it assigns exactly one truth value to every variable of $\mathcal{N}$. Let $C(\phi')$ be the set of clauses satisfied by ϕ'. We define $g : \mathcal{N} \times \{0,1\} \to \mathbb{R}^+$ using $g(\phi') \triangleq f(C(\phi'))$. Using this notation, we can restate SSAT as the problem of maximizing the function g over the set of feasible assignments. We need the following lemma.

LEMMA 5.1. *The function g is a normalized monotone submodular function.*

Proof. It is easy to see that g is normalized and monotone; however, proving it is also submodular requires some work. Consider two sets $A, B \subseteq \mathcal{N} \times \{0,1\}$. Using the submodularity and monotonicity of f, we get

$$\begin{aligned} g(A) + g(B) &= f(C(A)) + f(C(B)) \geq f(C(A) \cup C(B)) + f(C(A) \cap C(B)) \\ &\geq f(C(A \cup B)) + f(C(A \cap B)) = g(A \cup B) + g(A \cap B). \qquad \square \end{aligned}$$

The algorithm we design for SSAT conducts n iterations. It maintains at each iteration $1 \leq i \leq n$ two assignments X_i and Y_i which always satisfy $X_i \subseteq Y_i$. Initially, $X_0 = \emptyset$ assigns no truth values to the variables, and $Y_0 = \mathcal{N} \times \{0,1\}$ assigns both truth values to all variables. The algorithm considers the variables in an arbitrary order $u_1, u_2, \ldots, u_n$. For every variable u_i, the algorithm evaluates the marginal profit from assigning it only the truth value 0 in both assignments and assigning it only the truth value 1 in both assignments. Based on these marginal values, the algorithm makes a random decision on the truth value to be assigned to u_i. After the algorithm considers a variable u_i, both assignments X_i and Y_i agree on a single truth value for u_i. Thus, when the algorithm terminates both assignments are identical and feasible. A formal statement of the algorithm appears in Algorithm 4.

ALGORITHM 4. RANDOMIZED SSAT(f, Ψ).

```
1: $X_0 \leftarrow \emptyset$, $Y_0 \leftarrow \mathcal{N} \times \{0,1\}$.
2: for $i = 1$ to $n$ do
3:     $a_{i,0} \leftarrow g(X_{i-1} + (u_i, 0)) - g(X_{i-1})$, $a_{i,1} \leftarrow g(X_{i-1} + (u_i, 1)) - g(X_{i-1})$.
4:     $b_{i,0} \leftarrow g(Y_{i-1} - (u_i, 0)) - g(Y_{i-1})$, $b_{i,1} \leftarrow g(Y_{i-1} - (u_i, 1)) - g(Y_{i-1})$.
5:     $s_{i,0} \leftarrow \max\{a_{i,0} + b_{i,1}, 0\}$, $s_{i,1} \leftarrow \max\{a_{i,1} + b_{i,0}, 0\}$.
6:     with probability $s_{i,0}/(s_{i,0} + s_{i,1})^*$ do: $X_i \leftarrow X_{i-1} + (u_i, 0)$,
       $Y_i \leftarrow Y_{i-1} - (u_i, 1)$.
7:     else (with the complement probability $s_{i,1}/(s_{i,0} + s_{i,1})$) do:
8:     $X_i \leftarrow X_{i-1} + (u_i, 1)$, $Y_i \leftarrow Y_{i-1} - (u_i, 0)$.
9: return $X_n$ (or equivalently $Y_n$).
```

* If $s_{i,0} = s_{i,1} = 0$, we assume $s_{i,0}/(s_{i,0} + s_{i,1}) = 1$.

The proof of Theorem 1.3 uses ideas similar to those used in previous proofs in this paper; however, unlike for the previous algorithms, here, the fact that the algorithm runs in linear time requires a proof. The source of the difficulty is that we have an oracle access to f but use the function g in the algorithm. Therefore, we need to prove that we may implement all the oracle queries to g that are conducted during the execution of the algorithm in linear time. As a first step we state the following useful lemma.

LEMMA 5.2. *For every $1 \leq i \leq n$,*

$$\mathbb{E}[g(OPT_{i-1}) - g(OPT_i)] \leq \frac{1}{2} \cdot \mathbb{E}\left[(g(X_i) - g(X_{i-1}) + g(Y_i) - g(Y_{i-1})\right], \tag{5.1}$$

where expectations are taken over the random choices of the algorithm.

Before proving Lemma 5.2, let us show that Theorem 1.3 follows from it.

Proof of Theorem 1.3. The proof has two parts: bounding the approximation ratio of the algorithm and analyzing its time complexity.

Proof of the approximation ratio of the algorithm. Summing up Lemma 5.2 for every $1 \leq i \leq n$ we get

$$\sum_{i=1}^{n} \mathbb{E}[g(OPT_{i-1}) - g(OPT_i)] \leq \frac{1}{2} \cdot \sum_{i=1}^{n} \mathbb{E}[g(X_i) - g(X_{i-1}) + g(Y_i) - g(Y_{i-1})].$$

The above sum is telescopic. Collapsing it, we get

$$\begin{aligned}\mathbb{E}[g(OPT_0) - g(OPT_n)] &\leq \frac{1}{2} \cdot \mathbb{E}[g(X_n) - g(X_0) + g(Y_n) - g(Y_0)] \\ &\leq \frac{\mathbb{E}[g(X_n) + g(Y_n) - g(Y_0)]}{2}.\end{aligned}$$

Recalling the definitions of OPT_0 and OPT_n, we obtain that

$$\mathbb{E}[g(X_n)] = \mathbb{E}[g(Y_n)] \geq g(OPT)/2 + g(Y_0)/4.$$

The approximation ratio now follows from the observation that Y_0 satisfies all clauses of Ψ, and therefore, $g(Y_0) \geq g(OPT)$.

Proof of the linear time complexity of the algorithm. We explain here how to answer all oracle queries of the algorithm of the forms $g(Y_{i-1})$ and $g(Y_{i-1} - (u_i, v))$ in linear time. Using an analogous idea, one can also answer all oracle queries of the forms $g(X_{i-1})$ and $g(X_{i-1} + (u_i, v))$ in linear time.

The algorithm first preprocesses the clauses, creating the following simple data structure. For every variable u_i and truth value v, the algorithm creates a linked list containing all clauses that are satisfied by setting the truth value of u_i to be v. The number of lists is $2n$, where n is the number of variables, and they are kept in an array of that size. The total length of all lists is the total number of literals of Ψ which is $O(\Psi)$. Additionally, the algorithm maintains an array of counters C of size $|\mathcal{C}|$ (the number of clauses). For every clause we keep in C the number of literals that satisfy the clause under Y_{i-1}. Initially, the number of literals satisfied equals the number of literals in the clause. This array can also be constructed initially in linear time. As the array C indicates for each clause whether it is satisfied,[5] it can be used to make queries to f.

Next, in every iteration the algorithm has to answer three queries: $g(Y_{i-1})$, $g(Y_{i-1} - (u_i, 0))$, and $g(Y_{i-1} - (u_i, 1))$.

(i) A query of the form $g(Y_{i-1})$ is answered by making an oracle query $f(C)$.

(ii) A query of the form $g(Y_{i-1} - (u_i, v))$ is answered using the following steps:

(1) Scan the list of (u_i, v).

[5]A clause is satisfied if and only if its counter is at least 1.

(2) Decrease the counter (in the array C) of every clause satisfied by (u_i, v). Note that counters that become zero correspond to clauses that are satisfied in Y_{i-1} but are no longer satisfied in $Y_{i-1} - (u_i, v)$.
(3) Make an oracle query $f(C)$.
(4) Scan again the list of (u_i, v) and increase back the counters of the clauses that were decreased.

The final step of each iteration is replacing the assignment Y_{i-1} with a new assignment $Y_{i-1} - (u_i, v)$. To this end, the algorithm scans again the list of $\{u_i, v\}$ and decreases the counters of all the clauses satisfied by $\{u_i, v\}$. Notice that each list is processed at most twice: once for answering a query of the form $g(Y_{i-1} - (u_i, v))$ and once (possibly) to update Y_{i-1} to $Y_{i-1} - (u_i, v)$. Hence, the total processing time is proportional to the total size of the lists (which is linear in the length of Ψ). Finally, note that since C is maintained as an array, decreasing, increasing, and querying the counters can all be done in $O(1)$. ☐

It is left to prove Lemma 5.2.

Proof. Notice that it suffices to prove inequality (5.1) conditioned on any event of the form $X_{i-1} = S_{i-1}$, where $S_{i-1} \subseteq \{(u_1, *), (u_2, *), \ldots, (u_{i-1}, *)\}$, for which the probability that $X_{i-1} = S_{i-1}$ is nonzero (note that according to the algorithm's definition X_{i-1} contains exactly a single element of the form $(u_j, *)$ for every $1 \leq j \leq i-1$). Hence, fix such an event corresponding to a S_{i-1}. The rest of the proof implicitly assumes everything is conditioned on this event. Notice that due to the conditioning, the following quantities become constants:

(i) $Y_{i-1} = S_{i-1} \cup \{(u_i, 0), (u_i, 1), \ldots, (u_n, 0), (u_n, 1)\}$.
(ii) $OPT_{i-1} = (OPT \cup X_{i-1}) \cap Y_{i-1}$.
(iii) $a_{i,0}$, $a_{i,1}$, $b_{i,0}$, and $b_{i,1}$.

Moreover, by Lemma 2.1, $a_{i,0} + b_{i,0} \geq 0$ and $a_{i,1} + b_{i,1} \geq 0$. Thus, $s_{i,0} + s_{i,1} \geq 0$, and it cannot be that both $s_{i,0}, s_{i,1}$ are strictly less than zero. Hence, we only need to prove the lemma for the following three cases:

Case 1 *($\mathbf{s_{i,0} \geq 0}$ and $\mathbf{s_{i,1} \leq 0}$).* In this case $\frac{s_{i,0}}{s_{i,0}+s_{i,1}} = 1$, and the following always happen: $Y_i = Y_{i-1} - (u_i, 1)$ and $X_i \leftarrow X_{i-1} + (u_i, 0)$. Hence,

$$g(X_i) - g(X_{i-1}) + g(Y_i) - g(Y_{i-1}) = a_{i,0} + b_{i,1} = s_{i,0}.$$

By our definition $OPT_i = (OPT \cup X_i) \cap Y_i = OPT_{i-1} + (u_i, 0) - (u_i, 1)$. Thus, we are left to prove that

$$\begin{aligned} &g(OPT_{i-1}) - g(OPT_{i-1} + (u_i, 0) - (u_i, 1)) \\ &\quad \leq \frac{1}{2} \cdot [g(X_i) - g(X_{i-1}) + g(Y_i) - g(Y_{i-1})] = \frac{s_{i,0}}{2}. \end{aligned}$$

There are two cases. If $(u_i, 0) \in OPT, (u_i, 1) \notin OPT$, then the left-hand side of the last expression is 0, which is clearly no larger than the nonnegative $s_{i,0}/2$. If $(u_i, 0) \notin OPT, (u_i, 1) \in OPT$, then

$$\begin{aligned} g(OPT_{i-1}) - g(OPT_{i-1} + (u_i, 0) - (u_i, 1)) &= [g(OPT_{i-1}) - g(OPT_{i-1} + (u_i, 0))] \\ &\quad + [g(OPT_{i-1} + (u_i, 0)) - g(OPT_{i-1} + (u_i, 0) - (u_i, 1))] \\ &\leq g(Y_{i-1} - (u_i, 0)) - g(Y_{i-1}) + g(X_{i-1} + (u_i, 1)) - g(X_{i-1}) \\ &= s_{i,1} \leq 0 \leq s_{i,0}/2. \end{aligned}$$

The first inequality follows from submodularity since $OPT_{i-1} \subseteq Y_{i-1} - (u_i, 0)$ and $X_{i-1} \subseteq OPT_{i-1} + (u_i, 0) - (u_i, 1)$.

Case 2 ($\mathbf{s_{i,0} < 0}$ *and* $\mathbf{s_{i,1} \geq 0}$*).* This case is analogous to the previous one, and therefore we omit its proof.

Case 3 ($\mathbf{s_{i,0} \geq 0}$ *and* $\mathbf{s_{i,1} > 0}$*).* In this case $s_{i,0} = a_{i,0} + b_{i,1}$, $s_{i,1} = a_{i,1} + b_{i,0}$, and so

$$\begin{aligned}
&\mathbb{E}[g(X_i) - g(X_{i-1}) + g(Y_i) - g(Y_{i-1})] \\
&\quad = \frac{s_{i,0}}{s_{i,0} + s_{i,1}} \cdot [g(X_{i-1} + (u_i, 0)) - g(X_{i-1}) + g(Y_{i-1} - (u_i, 1)) - g(Y_{i-1})] \\
&\qquad + \frac{s_{i,1}}{s_{i,0} + s_{i,1}} \cdot [g(X_{i-1} + (u_i, 1)) - g(X_{i-1}) + g(Y_{i-1} - (u_i, 0)) - g(Y_{i-1})] \\
&\quad = \frac{s_{i,0}^2 + s_{i,1}^2}{s_{i,0} + s_{i,1}}.
\end{aligned} \tag{5.2}$$

Next, we upper bound $\mathbb{E}[g(OPT_{i-1}) - g(OPT_i)]$. As $OPT_i = (OPT \cup X_i) \cap Y_i$, and given the random choices of the algorithm

$$\begin{aligned}
\mathbb{E}[g(OPT_{i-1}) - g(OPT_i)] &= \frac{s_{i,0}}{s_{i,0} + s_{i,1}} \cdot [g(OPT_{i-1}) - g(OPT_{i-1} + (u_i, 0) - (u_i, 1))] \\
&\quad + \frac{s_{i,1}}{s_{i,0} + s_{i,1}} \cdot [g(OPT_{i-1}) - g(OPT_{i-1} + (u_i, 1) - (u_i, 0))] \\
&\leq \frac{s_{i,0} s_{i,1}}{s_{i,0} + s_{i,1}}.
\end{aligned} \tag{5.3}$$

The final inequality follows by considering two cases. Note first that $\{(u_i, 0), (u_i, 1)\} \subseteq Y_{i-1}$ and $\{(u_i, 0), (u_i, 1)\} \cap X_{i-1} = \varnothing$. If $(u_i, 0) \notin OPT_{i-1}$, and $(u_i, 1) \in OPT_{i-1}$, then the second term of the left-hand side of the last inequality is zero. Moreover, $OPT_{i-1} = ((OPT \cup X_{i-1}) \cap Y_{i-1}) \subseteq Y_{i-1} - (u_i, 0)$. Thus, by submodularity,

$$\begin{aligned}
&g(OPT_{i-1}) - g(OPT_{i-1} + (u_i, 0) - (u_i, 1)) \\
&\quad \leq g(X_{i-1} + (u_i, 1)) - g(X_{i-1}) + g(Y_{i-1} - (u_i, 0)) - g(Y_{i-1}) = s_{i,1}.
\end{aligned}$$

The other case is analogous. By (5.2) and (5.3), inequality (5.1) holds if

$$\frac{s_{i,0} s_{i,1}}{s_{i,0} + s_{i,1}} \leq \frac{1}{2} \cdot \left(\frac{s_{i,0}^2 + s_{i,1}^2}{s_{i,0} + s_{i,1}} \right),$$

which can be easily verified. ∎

A note on Max-SAT*.* The well known Max-SAT problem is in fact a special case of SSAT where f is a linear function. We note that Algorithm 4 can be applied to Max-SAT in order to achieve a (3/4)-approximation in linear time; however, this is not immediate. This result is summarized in the following theorem.

THEOREM 5.3. *Algorithm 4 has a linear time implementation for instances of Max-SAT.*

Proof. Consider the way oracle queries of g are answered in the proof of Theorem 1.3. The only use of queries to f made by this proof is in order to evaluate $f(C)$—the total weight of the clauses in the set C. In order to avoid these queries, one can use an additional counter $w(C)$ which holds the total weight of the clauses that are satisfied in C. Initially, this counter is the total weight of clauses in the formula. The counter can be updated in constant time along with the update of the array C. Whenever a counter of a clause in C becomes zero, the additional counter is decreased by the weight of the clause. If the counter of a clause is increased back from zero the weight of the clause is added to the counter. These additional updates are done in constant time. Answering oracle calls to $f(C)$ are done in constant time by returning the value of the counter. Hence, the total running time is linear in the size of the formula. $\square$

5.2. A linear time tight (3/4)-approximation for SW with two players. An input for SW consists of a ground set $\mathcal{N}$ of n elements and k players, each equipped with a normalized monotone submodular utility function $f_i : 2^{\mathcal{N}} \to \mathbb{R}^+$. The goal is to partition the elements among the players while maximizing the social welfare. Formally, the objective is to partition $\mathcal{N}$ into $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$ while maximizing $\sum_{i=1}^{k} f_i(\mathcal{N}_i)$.

We give below two different short proofs of Theorem 1.4 via reductions to SSAT and USM, respectively. The second proof is due to Vondrák [39].

Proof of Theorem 1.4. We provide here two proofs.

First Proof. Given an instance of SW with two players, construct an instance of SSAT as follows:

1. The set of variables is $\mathcal{N}$.
2. The CNF formula Ψ consists of $2|\mathcal{N}|$ singleton clauses, one for every possible literal.
3. The objective function $f : 2^{\mathcal{C}} \to \mathbb{R}^+$ is defined as follows. Let $P \subseteq \mathcal{C}$ be the set of clauses of Ψ consisting of positive literals. Then, $f(C) = f_1(C \cap P) + f_2(C \setminus P)$.

Every assignment ϕ to this instance of SSAT corresponds to a solution of SW using the following rule: $\mathcal{N}_1 = \{u \in \mathcal{N} \mid \phi(u) = 1\}$ and $\mathcal{N}_2 = \{u \in \mathcal{N} \mid \phi(u) = 0\}$. One can easily observe that this correspondence is reversible and that each assignment has the same value as the solution it corresponds to. Hence, the above reduction preserves approximation ratios.

Moreover, queries of f can be answered in constant time using the following technique. We track for every subset $C \subseteq \mathcal{C}$ in the algorithm the subsets $C \cap P$ and $C \setminus P$. For Algorithm 4 this can be done without affecting its running time. Then, whenever the value of $f(C)$ is queried, answering it simply requires making two oracle queries: $f_1(C \cap P)$ and $f_2(C \setminus P)$.

Second Proof. In any feasible solution to SW with two players, the set $\mathcal{N}_1$ uniquely determines the set $\mathcal{N}_2 = \mathcal{N} \setminus \mathcal{N}_1$. Hence, the value of the solution as a function of $\mathcal{N}_1$ is given by $g(\mathcal{N}_1) = f_1(\mathcal{N}_1) + f_2(\mathcal{N} \setminus \mathcal{N}_1)$. Thus, SW with two players can be restated as the problem of maximizing the function g over the subsets of $\mathcal{N}$.

Observe that the function g is a submodular function, but unlike f_1 and f_2, it is possibly nonmonotone. Moreover, we can answer queries to the function g using only two oracle queries to f_1 and f_2.[6] Thus, we obtain an instance of USM. We apply Algorithm 2 to this instance. Using the analysis of Algorithm 2, as is, provides only a (1/2)-approximation for our problem. However, by noticing that $g(\varnothing) + g(\mathcal{N}) \geq f_1(\mathcal{N}) + f_2(\mathcal{N}) \geq g(OPT)$, and plugging this into the analysis, the claimed (3/4)-approximation is obtained. $\square$

6. Discussion. The main result of this paper is a randomized linear time (1/2)-approximation algorithm for USM. This result is the best one can hope for assuming

[6]For every algorithm, assuming a representation of sets allowing addition and removal of a single element at a time, one can maintain the complement sets of all sets maintained by the algorithm without changing the time complexity. Hence, we need not worry about the calculation of $\mathcal{N} \setminus \mathcal{N}_1$.

the function f is accessed via a value oracle. We leave several interesting open questions for future research. The first question is whether it is possible to derandomize the algorithm to obtain a deterministic algorithm. The deterministic algorithm we presented provides only (1/3)-approximation. This is currently the best deterministic algorithm for USM (a deterministic, but slower, algorithm achieving $(1/3 - o(1))$-approximation was previously described by [12]). So far, there is no proof that derandomization is not possible. However, we suspect that no polynomial time deterministic algorithm can achieve (1/2)-approximation. A recent work by [24] shows that a large family of deterministic algorithms generalizing our algorithm fail to achieve (1/2)-approximation. Understanding the role of randomization in this basic problem is an interesting avenue for further research.

A second interesting direction is extending our basic algorithm to constrained variants of the submodular maximization problem. Many such variants were studied by previous works. Applying our algorithm to constrained problems is difficult since it provides no guarantees on the set of elements chosen, nor on the number of such elements. Recently, Buchbinder et al. [3] managed to extend the fractional version of the algorithm to handle cardinality constraints. Their algorithm provides (1/2)-approximation when the algorithm is constrained to choose at most (or exactly) $k = n/2$ elements. However, its proven performance guarantee deteriorates as the cardinality constraint k decreases. In our opinion, the algorithm presented by [3] (or a similar variant) should achieve a constant approximation also for small values of k. Extending our algorithm to other constraint types remains an intriguing open problem.

REFERENCES

[1] A. A. Ageev and M. I. Sviridenko, *An* 0.828 *approximation algorithm for the uncapacitated facility location problem*, Discrete Appl. Math., 93 (1999), pp. 149–156.

[2] S. Ahmed and A. Atamtürk, *Maximizing a class of submodular utility functions*, Math. Program., 128 (2011), pp. 149–169.

[3] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz, *Submodular maximization with cardinality constraints*, in Proceedings of SODA, 2014, pp. 1433–1452.

[4] G. Calinescu, C. Chekuri, M. Pal, and J. Vondrák, *Maximizing a monotone submodular function subject to a matroid constraint*, SIAM J. Comput., 40 (2011), pp. 1740–1766.

[5] C. Chekuri, J. Vondrák, and R. Zenklusen, *Dependent randomized rounding via exchange properties of combinatorial structures*, in Proceedings of FOCS, 2010, pp. 575–584.

[6] C. Chekuri, J. Vondrák, and R. Zenklusen, *Submodular function maximization via the multilinear relaxation and contention resolution schemes*, in Proceedings of STOC, 2011, pp. 783–792.

[7] V. P. Cherenin, *Solving some combinatoiral problems of optimal planning by the method of successive calculations (in Russian)*, in Proceedings of the Conference on Experiences and Perspectives on the Applications of Mathematical Methods and Electronic Computers in Planning, Novosibirsk, 1962.

[8] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser, *Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms*, Management Sci., 23 (1977), pp. 789–810.

[9] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser, *On the uncapacitated location problem*, Ann. Discrete Math., 1 (1977), pp. 163–177.

[10] S. Dobzinski and J. Vondrák, *From query complexity to computational complexity*, in Proceedings of STOC, 2012, pp. 1107–1116.

[11] S. Dughmi, T. Roughgarden, and M. Sundararajan, *Revenue submodularity*, Theory Comput., 8 (2012), pp. 95–119.

[12] U. Feige, V. S. Mirrokni, and J. Vondrák, *Maximizing non-monotone submodular functions*, SIAM J. Comput., 40 (2011), pp. 1133–1153.

[13] M. Feldman, J. Naor, and R. Schwartz, *Nonmonotone submodular maximization via a structural continuous greedy algorithm*, in Proceedings of ICALP, 2011, pp. 342–353.

[14] M. Feldman, J. Naor, and R. Schwartz, *A unified continuous greedy algorithm for submodular maximization*, in Proceedings of FOCS, 2011, pp. 570–579.

[15] S. O. Gharan and J. Vondrák, *Submodular maximization by simulated annealing*, in Proceedings of SODA, 2011, pp. 1098–1117.

[16] M. X. Goemans and D. P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM, 42 (1995), pp. 1115–1145.

[17] B. Goldengorin and D. Ghosh, *A multilevel search algorithm for the maximization of submodular functions applied to the quadratic cost partition problem*, J. Global Optim., 32 (2005), pp. 65–82.

[18] B. Goldengorin, G. Sierksma, G. A. Tijssen, and M. Tso, *The data-correcting algorithm for the minimization of supermodular functions*, Management Sci., 45 (1999), pp. 1539–1551.

[19] B. Goldengorin, G. A. Tijssen, and M. Tso, *The Maximization of Submodular Functions: Old and New Proofs for the Correctness of the Dichotomy Algorithm*, Research report 99A17, Research Institute SOM (Systems, Organisations and Management), University of Groningen, 1999.

[20] A. Gupta, A. Roth, G. Schoenebeck, and K. Talwar, *Constrained non-monotone submodular maximization: Offline and secretary algorithms*, in Internet and Network Economics, Lecture Notes in Comput. Sci. 6484, Springer-Verlag, New York, 2010, pp. 246–257.

[21] E. Halperin and U. Zwick, *Combinatorial approximation algorithms for the maximum directed cut problem*, in Proceedings of SODA, 2001, pp. 1–7.

[22] J. Hartline, V. Mirrokni, and M. Sundararajan, *Optimal marketing strategies over social networks*, in Proceedings of WWW, 2008, pp. 189–198.

[23] J. Håstad, *Some optimal inapproximability results*, J. ACM, 48 (2001), pp. 798–859.

[24] N. Huang and A. Borodin, *Bounds on Double-Sided Myopic Algorithms for Unconstrained Non-Monotone Submodular Maximization*, CoRR abs/1312.2173, 2013.

[25] S. Iwata and K. Nagano, *Submodular function minimization under covering constraints*, in Proceedings of FOCS, 2009, pp. 671–680.

[26] S. Jegelka and J. Bilmes, *Cooperative Cuts: Graph Cuts with Submodular Edge Weights*, Technical report 189-03-2010, Max Planck Institute for Biological Cybernetics, Tuebingen, 2010.

[27] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[28] V. R. Khachaturov, *Some Problems of the Consecutive Calculation Method and Its Applications to Location Problems (in Russian)*, Ph.D. thesis, Central Economics and Mathematics Institute, Russian Academy of Sciences, Moscow, 1968.

[29] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell, *Optimal inapproximability results for max-cut and other* 2*-variable csps?*, SIAM J. Comput., 37 (2007), pp. 319–357.

[30] A. Kulik, H. Shachnai, and T. Tamir, *Maximizing submodular set functions subject to multiple linear constraints*, in Proceedings of SODA, 2009, pp. 545–554.

[31] H. Lee, G. L. Nemhauser, and Y. Wang, *Maximizing a submodular function by integer programming: Polyhedral results for the quadratic case*, European J. Oper. Res., 94 (1996), pp. 154–166.

[32] J. Lee, V. S. Mirrokni, V. Nagarajan, and M. Sviridenko, *Maximizing non-monotone submodular functions under matroid or knapsack constraints*, SIAM J. Discrete Math., 23 (2010), pp. 2053–2078.

[33] J. Lee, M. Sviridenko, and J. Vondrák, *Submodular maximization over multiple matroids via generalized exchange properties*, in Proceedings of APPROX, 2009, pp. 244–257.

[34] L. Lovász, M. Grötschel, and A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatoria, 1 (1981), pp. 169–197.

[35] M. Minoux, *Accelerated greedy algorithms for maximizing submodular set functions*, in Optimization Techniques, J. Stoer, ed., Lecture Notes in Control and Inform. Sci. 7, Springer, Berlin, 1978, pp. 234–243.

[36] A. S. Schulz and N. A. Uhan, *Approximating the least core and least core value of cooperative games with supermodular costs*, Discrete Optim., 10 (2013), pp. 163–180.

[37] Z. Svitkina and L. Fleischer, *Submodular approximation: Sampling-based algorithms and lower bounds*, in Proceedings of FOCS, 2008, pp. 697–706.

[38] L. Trevisan, G. B. Sorkin, M. Sudan, and D. P. Williamson, *Gadgets, approximation, and linear programming*, SIAM J. Comput., 29 (2000), pp. 2074–2097.
[39] J. Vondrák, Personal communication, 2012.
[40] J. Vondrák, *Symmetry and approximability of submodular maximization problems*, in Proceedings of FOCS, 2009, pp. 651–670.