# Towards Nearly-Linear Time Algorithms for Submodular Maximization with a Matroid Constraint

**Alina Ene**
Department of Computer Science, Boston University, MA, USA
aene@bu.edu

**Huy L. Nguyen**
College of Computer and Information Science, Northeastern University, Boston, MA, USA
hlnguyen@cs.princeton.edu

**Abstract**

We consider fast algorithms for monotone submodular maximization subject to a matroid constraint. We assume that the matroid is given as input in an explicit form, and the goal is to obtain the best possible running times for important matroids. We develop a new algorithm for a *general matroid constraint* with a $1 - 1/e - \epsilon$ approximation that achieves a fast running time provided we have a fast data structure for maintaining an approximately maximum weight base in the matroid through a sequence of decrease weight operations. We construct such data structures for graphic matroids and partition matroids, and we obtain the *first algorithms* for these classes of matroids that achieve a nearly-optimal, $1 - 1/e - \epsilon$ approximation, using a nearly-linear number of function evaluations and arithmetic operations.

**2012 ACM Subject Classification** Theory of computation → Submodular optimization and polymatroids

**Keywords and phrases** submodular maximization, matroid constraints, fast running times

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2019.54

**Category** Track A: Algorithms, Complexity and Games

**Related Version** `https://arxiv.org/abs/1811.07464`

**Funding** *Alina Ene*: Partially supported by NSF CAREER grant CCF-1750333 and NSF grant CCF-1718342.
*Huy L. Nguyen*: Partially supported by NSF CAREER grant CCF-1750716.

**Acknowledgements** This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing.

## 1 Introduction

In this paper, we consider fast algorithms for monotone submodular maximization subject to a matroid constraint. Submodular maximization is a central problem in combinatorial optimization that captures several problems of interest, such as maximum coverage, facility location, and welfare maximization. The study of this problem dates back to the seminal work of Nemhauser, Wolsey and Fisher from the 1970's [20, 21, 12]. Nemhauser *et al.* introduced a very natural Greedy algorithm for the problem that iteratively builds a solution by selecting the item with the largest marginal gain on top of previously selected items, and they showed that this algorithm achieves a $1 - 1/e$ approximation for a cardinality constraint and a $1/2$ approximation for a general matroid constraint. The maximum coverage problem is a special case of monotone submodular maximization with a cardinality constraint and it is $1 - 1/e$

hard to approximate [10], and thus the former result is optimal. Therefore the main question that was left open by the work of Nemhauser *et al.* is whether one can obtain an optimal, $1 - 1/e$ approximation, for a general matroid constraint.

In a celebrated line of work [6, 24], Calinescu *et al.* developed a framework based on continuous optimization and rounding that led to an optimal $1 - 1/e$ approximation for the problem. The approach is to turn the discrete optimization problem of maximizing a submodular function $f$ subject to a matroid constraint into a continuous optimization problem where the goal is to maximize the multilinear extension $F$ of $f$ (a continuous function that extends $f$) subject to the matroid polytope (a convex polytope whose vertices are the feasible integral solutions). The continuous optimization problem can be solved approximately within a $1 - 1/e$ factor using a continuous Greedy algorithm [24] and the resulting fractional solution can be rounded to an integral solution without any loss [1, 6, 8]. The resulting algorithm achieves the optimal $1 - 1/e$ approximation in polynomial time.

Unfortunately, a significant drawback of this approach is that it leads to very high running times. Obtaining fast running times is a fundamental direction both in theory and in practice, due to the numerous applications of submodular maximization in machine learning, data mining, and economics [18, 17, 14, 16, 9]. This direction has received considerable attention [2, 11, 3, 19, 5, 7], but it remains a significant challenge for almost all matroid constraints.

Before discussing these challenges, let us first address the important questions on how the input is represented and how we measure running time. The algorithms in this paper as well as prior work assume that the submodular function is represented as a value oracle that takes as input a set $S$ and returns $f(S)$. For all these algorithms, the number of calls to the value oracle for $f$ dominates the running time of the algorithm (up to a logarithmic factor), and thus we assume for simplicity that each call takes constant time.

The algorithms fall into two categories with respect to how the matroid is represented: the *independence oracle* algorithms assume that the matroid is represented using an oracle that takes as input a set $S$ and returns whether $S$ is feasible (independent); the *representable matroid* algorithms assume that the matroid is given as input in an explicit form. The representable matroid algorithms can be used for only a subclass of matroids, namely those that can be represented as a linear matroid over vectors in some field[1], but this class includes practically-relevant matroids such as the uniform, partition, laminar, graphical, and general linear matroids. The oracle algorithms apply to all matroids, but they are *unlikely to lead to the fastest possible running times:* even an ideal algorithm that makes only $O(k)$ independence calls has a running time that is $\Omega(k^2)$ in the independence oracle model (each oracle call needs to read its input, which takes $\Theta(k)$ time in the worst case), even if the matroid is a representable matroid such as a partition or a graphic matroid. This is because Thus there have always been parallel lines of research for representable matroids and general matroids.

This work falls in the first category, i.e., we assume that the matroid is given as input in an explicit form, and the goal is to obtain the best possible running times. Note that, although all the representable matroids are linear matroids, it is necessary to consider each class separately, since they have very different running times to even verify if a given solution is feasible: for simple explicit matroids such as a partition or a graphic matroid, checking whether a solution is feasible takes $O(n)$ time, where $n$ is the size of the ground set of the matroid; for general explicit matroids represented using vectors in some field, checking whether a solution is feasible takes $O(k^\omega)$ time, where $k$ is the rank of the matroid and $\omega$ is the exponent for fast matrix multiplication.

[1] In a linear matroid, the ground set is a collection of $n$ vectors and a subset of the vectors is feasible (independent) if the vectors are linearly independent.

Since in many practical settings only nearly-linear running times are feasible, an important question to address is:

*For which matroid constraints can we obtain*
*a nearly-optimal* $1 - 1/e - \epsilon$ *approximation in nearly-linear time?*

Prior to this work, the only example of such a constraint was a cardinality constraint. For a partition matroid constraint, the fastest running time is $\Omega(n^{3/2})$ in the worst case when $k = \Omega(n)$ [5]. For a graphical matroid constraint, no faster algorithms are known than a general matroid, and the running time is $\Omega(n^2)$ in the worst case when $k = \Omega(n)$ [3]. Obtaining a best-possible, nearly-linear running time has been very challenging even for these classes of matroids for the following reasons:

**The continuous optimization is a significant time bottleneck.** The continuous optimization problem of maximizing the multilinear extension subject to the matroid polytope is an integral component in all algorithms that achieve a nearly-optimal approximation guarantee. However, the multilinear extension is expensive to evaluate even approximately. To achieve the nearly-optimal approximation guarantees, the evaluation error needs to be very small and in a lot of cases, the error needs to be $O(n^{-1})$ times the function value. As a result, a single evaluation of the multilinear extension requires $\Omega(n)$ evaluations of $f$. Thus, even a very efficient algorithm with $O(n)$ queries to the multilinear extension would require $\Omega(n^2)$ running time.

**Rounding the fractional solution is a significant time bottleneck as well.** Consider a matroid constraint of rank $k$. The fastest known rounding algorithm is the swap rounding, which requires $k$ swap operations: in each operation, the algorithm has two bases $B_1$ and $B_2$ and needs to find $x \in B_1, y \in B_2$ such that $B_1 \setminus \{x\} \cup \{y\}$ and $B_2 \setminus \{y\} \cup \{x\}$ are bases. The typical implementation is to pick some $x \in B_1$ and try all $y$ in $B_2$, which requires us to check independence for $k$ solutions. Thus, overall, the rounding algorithm checks independence for $\Omega(k^2)$ solutions. Furthermore, each feasibility check takes $\Omega(k)$ time just to read the input. Thus a generic rounding for a matroid takes $\Omega(k^3)$ time.

Thus, in order to achieve a fast overall running time, one needs fast algorithms for both the continuous optimization and the rounding. In this work, we provide such algorithms for partition and graphic matroids, and we obtain the first algorithms with nearly-linear running times. At the heart of our approach is a general, nearly-linear time reduction that reduces the submodular maximization problem to two data structure problems: maintain an approximately maximum weight base in the matroid through a sequence of decrease-weight operations, and maintain an independent set in the matroid that allows us to check whether an element can be feasibly added. This reduction applies to any representable matroid, and thus it opens the possibility of obtaining faster running times for other classes of matroids.

## 1.1 Our contributions

We now give a more precise description of our contributions. We develop a new algorithm for maximizing the multilinear extension subject to a *general matroid constraint* with a $1 - 1/e - \epsilon$ approximation that achieves a fast running time provided we have fast data structures with the following functionality:

- *A maximum weight base data structure:* each element has a weight, and the goal is to maintain an approximately maximum weight base in the matroid through a sequence of operations, where each operation can only decrease the weight of a single element;

- *An independent set data structure* that maintains an independent set in the matroid and supports two operations: add an element to the independent set, and check whether an element can be added to the independent set while maintaining independence.

▶ **Theorem 1.** *Let $f$ be a monotone submodular function and let $\mathcal{M}$ be a matroid on a ground set of size $n$. Let $F$ be the multilinear extension of $f$ and $P(\mathcal{M})$ be the matroid polytope of $\mathcal{M}$. Suppose that we have a data structure for maintaining a maximum weight base and independent set as described above. There is an algorithm for the problem* $\max_{x \in P(\mathcal{M})} F(x)$ *that achieves a $1 - 1/e - \epsilon$ approximation using $O(n\ \mathrm{poly}(\log n, 1/\epsilon))$ calls to the value oracle for $f$, data structure operations, and additional arithmetic operations.*

Using our continuous optimization algorithm and additional results that are included in the full version of this paper, we obtain the first nearly-linear time algorithms for both the discrete and continuous problem with a graphic and a partition matroid constraint. In the graphic matroid case, the maximum weight base data structure is a dynamic maximum weight spanning tree (MST) data structure and the independent data structure is a dynamic connectivity data structure (e.g., union-find), and we can use existing data structures that guarantee a poly-logarithmic amortized time per operation [15, 13, 23]. For a partition matroid, we provide data structures with a constant amortized time per operation. We also address the rounding step and provide a nearly-linear time algorithm for rounding a fractional solution in a graphic matroid. A nearly-linear time rounding algorithm for a partition matroid was provided in [5].

▶ **Theorem 2.** *There is an algorithm for maximizing a monotone submodular function subject to a generalized partition matroid constraint that achieves a $1 - 1/e - \epsilon$ approximation using $O(n\ \mathrm{poly}(1/\epsilon, \log n))$ function evaluations and arithmetic operations.*

▶ **Theorem 3.** *There is an algorithm for maximizing a monotone submodular function subject to a graphic matroid constraint that achieves a $1 - 1/e - \epsilon$ approximation using $O(n\ \mathrm{poly}(1/\epsilon, \log n))$ function evaluations and arithmetic operations.*

Previously, the best running time for a partition matroid was $\Omega(n^{3/2}\ \mathrm{poly}(1/\epsilon, \log n))$ in the worst case when $k = \Omega(n)$ [5]. The previous best running time for a graphic matroid is the same as the general matroid case, which is $\Omega(n^2\ \mathrm{poly}(1/\epsilon, \log n))$ in the worst case when $k = \Omega(n)$ [3].

As shown by Vondrak [24], there is an efficient reduction from the *submodular welfare maximization* problem to the submodular maximization problem with a partition matroid constraint. Using this reduction and our algorithm for a partition matroid, we obtain a nearly-linear time algorithm for welfare maximization as well.

▶ **Theorem 4.** *There is a $1 - 1/e - \epsilon$ approximation algorithm for submodular welfare maximization using $O(n\mathrm{poly}(1/\epsilon, \log n))$ function evaluations and arithmetic operations.*

We conclude with a formal statement of the contributions made in this paper on which the results above are based.

▶ **Theorem 5.** *There is a dynamic data structure for maintaining a maximum weight base in a partition matroid through a sequence of decrease weight operations with an $O(1)$ amortized time per operation.*

▶ **Theorem 6.** *There is a randomized algorithm based on swap rounding for the graphic matroid polytope that takes as input a point $x$ represented as a convex combination of bases and rounds it to an integral solution $S$ such that $\mathbb{E}[f(S)] \geq F(x)$. The running time of the algorithm is $O(nt \log^2 n)$, where $t$ is the number of bases in the convex combination of $x$.*

## 1.2 Technical overview

The starting point of our approach is the work [5]. They observed that the running time of the continuous algorithm using the multilinear extension of [3] depends on the value of the maximum weight base when the value is measured in the modular approximation $f'(S) = \sum_{e \in S} f(e)$. It is clear that this approximation is at least the original function and it can be much larger. They observed that the running time is proportional to the ratio between the maximum weight base when weights are measured using the modular approximation compared with the optimal solution when weights are measured using the original function. On the other hand, in the greedy algorithm, the gain in every greedy step is proportional to the maximum weight base when weights are measured using the modular approximation. Thus, the discrete greedy algorithm makes fast progress precisely when the continuous algorithm is slow and vice versa. Therefore, one can start with the discrete greedy algorithm and switch to the continuous algorithm when the maximum weight solution is small even when weights are measured using the modular approximation.

Our algorithm consists of two key components: (1) a fast dynamic data structure for maintaining an approximate maximum weight base through a sequence of greedy steps, and (2) an algorithm that makes only a small number of queries to the data structure. Even if fast dynamic data structures are available, previous algorithms including that of [5] cannot achieve a fast time, since they require $\Omega(nk)$ queries to the data structure: the algorithm of [5] maintains the marginal gain for every element in the current base and it updates them after each greedy step; since each greedy step changes the marginal gain of every element in the base, this approach necessitates $\Omega(k)$ data structure queries per greedy step.

Our new approach uses random sampling to ensure that the number of queries to the data structure is nearly-linear. After each greedy step, our algorithm randomly samples elements from the base to check and update the marginal gains. Because of sampling, it can only ensure that at least $1/2$ of the elements in every value range have good estimates of their values. However, this is sufficient for maintaining an approximate maximum weight base. The benefit is that the running time becomes much faster: the number of checks that do not result in updates is small and if we make sure that an update only happens when the marginal gain change by a factor $1 - \epsilon$ then the total number of updates is at most $O(n \log n/\epsilon)$. Thus we obtain an algorithm with only a nearly-linear number of data structure queries and additional running time for *any matroid constraint*.

Our approach reduces the algorithmic problem to the data structure problem of maintaining an approximate maximum weight base through a sequence of value updates. In fact, the updates are only decrement in the values and thus can utilize even the decremental data structures as opposed to fully dynamic ones. In the case of a partition matroid constraint, one can develop a simple ad-hoc solution. In the case of a graphic matroid, one can use classical data structures for maintaining minimum spanning trees [15].

In both cases, fast rounding algorithms are also needed. The work [5] gives an algorithm for the partition matroid. We give an algorithm for the graphic matroid based on swap rounding and classical dynamic graph data structures. To obtain fast running time, in each rounding step, instead of swapping a generic pair, we choose a pair involving a leaf of the spanning tree.

## 1.3 Basic definitions and notation

**Submodular functions.** Let $f : 2^V \to \mathbb{R}_+$ be a set function on a finite ground set $V$ of size $n := |V|$. The function is *submodular* if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for all subsets $A, B \subseteq V$. The function is *monotone* if $f(A) \leq f(B)$ for all subsets $A \subseteq B \subseteq V$. We assume that the function $f$ is given as a value oracle that takes as input any set $S \subseteq V$ and returns $f(S)$. We let $F : [0,1]^V \to \mathbb{R}_+$ denote the multilinear extension $f$. For every $x \in [0,1]^V$, we have

$$F(x) = \sum_{S \subseteq V} f(S) \prod_{e \in S} x_e \prod_{e \notin S} (1 - x_e) = \mathbb{E}[R(x)],$$

where $R(x)$ is a random set that includes each element $e \in V$ independently with probability $x_e$.

**Matroids.** A matroid $\mathcal{M} = (V, \mathcal{I})$ on a ground set $V$ is a collection $\mathcal{I}$ of subsets of $V$, called independent sets, that satisfy certain properties (see e.g., [22], Chapter 39). In this paper, we consider matroids that are given to the input to the algorithm. Of particular interest are the partition and graphic matroids. A generalized partition matroid is defined as follows. We are given a partition $V_1, V_2, \ldots, V_h$ of $V$ into disjoint subsets and budgets $k_1, k_2, \ldots, k_h$. A set $S$ is independent ($S \in \mathcal{I}$) if $|S \cap V_i| \leq k_i$ for all $i \in [h]$. We let $k = \sum_{i=1}^{h} k_i$ denote the rank of the matroid. A graphic matroid is defined as follows. We are given a connected graph on $k+1$ vertices and $n$ edges. The independent sets of the matroid are the forests of this graph.

**Additional notation.** Given a set $S \in \mathcal{I}$, we let $f_S$ denote the function $f_S : 2^{V \setminus S} \to \mathbb{R}_{\geq 0}$ such that $f_S(S') = f(S' \cup S) - f(S)$ for all $S' \subseteq V \setminus S$. We let $\mathcal{M}/S = (V \setminus S, \mathcal{I}')$ denote the matroid obtained by contracting $S$ in $\mathcal{M}$, i.e., $S' \in \mathcal{I}'$ iff $S' \cup S \in \mathcal{I}$. We let $P(\mathcal{M})$ denote the matroid polytope of $\mathcal{M}$: $P(\mathcal{M})$ is the convex hull of the indicator vectors of the bases of $\mathcal{M}$, where a base is an independent set of maximum size.

**Constant factor approximation to $f(\mathrm{OPT})$.** Our algorithm needs a $O(1)$ approximation to $f(\mathrm{OPT})$. Such an approximation can be computed very efficiently (see e.g. [5], Lemma 3.2).

## 1.4 Paper organization

In Section 2, we describe our algorithm for the continuous optimization problem of maximizing the multilinear extension subject to a general matroid constraint, with the properties stated in Theorem 1. As discussed in the introduction, our algorithm uses certain data structures to achieve a fast running time. In the full version of the paper, we show how to obtain these data structures for partition and graphic matroids. By combining these data structures with the results of Section 2, we obtain nearly-linear time algorithms for the continuous problem of maximizing the multilinear extension subject to a partition and graphic matroid constraint. To obtain a fast algorithm for the discrete problem, we also need a fast algorithm to round the fractional solution. Buchbinder *et al.* [5] give a nearly-linear time rounding algorithm for a partition matroid. In the full version of the paper, we give a nearly-linear time rounding algorithm for a graphic matroid, and prove Theorem 6. These results together give Theorems 2 and 3.

## 2 The algorithm for the continuous optimization problem

In this section, we describe and analyze our algorithm for the problem $\max_{x \in P(\mathcal{M})} F(x)$ for a general matroid $\mathcal{M}$, and prove Theorem 1. The algorithm is given in Algorithm 1 and it combines the continuous Greedy algorithm of [3] with a discrete Greedy algorithm that we provide in this paper, building on [5].

**Algorithm 1** Algorithm for the continuous problem $\max_{x \in P(\mathcal{M})} F(x)$.

```
1: procedure ContinuousMatroid($f, \mathcal{M}, \epsilon$)
2:     $c' = \Theta(1/\epsilon)$, where the $\Theta$ hides a sufficiently large absolute constant
3:     $S$ = LazySamplingGreedy($f, \mathcal{M}, \epsilon$)
4:     $x$ = ContinuousGreedy($f_S, \mathcal{M}/S, c', \epsilon$)
5:     return $\mathbf{1}_S \vee x$    ⟨⟨ $x \vee y$ is the vector $(x \vee y)_i = \max\{x_i, y_i\}$ for all $i$ ⟩⟩
6: end procedure
```

**The continuous Greedy algorithm.** The algorithm used on line 4 is the algorithm of [3]. To obtain a fast running time, we use an independent set data structure to maintain the independent sets constructed by the algorithm. The data structure needs to support two operations: add an element to the independent set, and check whether an element can be added to the independent set while maintaining independence. For a partition matroid, such a data structure with $O(1)$ time per operation is trivial to obtain. For a graphic matroid, we can use a union-find data structure [13, 23] with a $O(\log^* k)$ amortized time per operation.

▶ **Lemma 7** (Corollary 3.1 in [5]; [3]). *When run with values $c$ and $\delta$ as input,* ContinuousGreedy *uses $O(n \ln(n/\delta)/\delta^2)$ independent set data structure operations, and $O(cn \ln^2(n/\delta)/\delta^4)$ queries to the value oracle of $f$ and additional arithmetic operations. Moreover, if $\max_{S \in \mathcal{I}} \sum_{e \in S} f(e) \leq c \cdot f(\mathrm{OPT})$, where $\mathrm{OPT} \in \mathrm{argmax}_{S \in \mathcal{I}} f(S)$, the solution $x$ returned by the algorithm satisfies $F(x) \geq (1 - \frac{1}{e} - \delta) f(\mathrm{OPT})$.*

The discrete Greedy algorithm is given in Algorithm 2. The algorithm works for any matroid constraint for which we can provide a fast data structure for maintaining a maximum weight base (note that the base is only an approximate maximum weight base, and we drop the word approximate for simplicity). We now describe the properties we require from this data structure. As discussed in the introduction, we give such data structures for a graphic matroid and a partition matroid in the full version of this paper.

**The dynamic maximum weight base data structure.** Algorithm 2 makes use of a data structure for maintaining the maximum weight base in the matroid, where each element has a weight and the weights are updated through a sequence of updates that can only decrease the weights. The data structure needs to support the following operation: UpdateBase decreases the weight of an element and it updates the base to a maximum weight base for the updated weights. The data structures that we provide in the full version of the paper for a graphic and a partition matroid support this operation in $O(\mathrm{poly}(\log k))$ amortized time.

We note here that the data structure maintains a maximum weight base of the original matroid $\mathcal{M}$, and not the contracted matroid $\mathcal{M}/S$ obtained after picking a set $S$ of elements. This suffices for us, since the discrete Greedy algorithm that we use will not change the weight of an element after it was added to the solution $S$. Due to this invariant, we can show that the maximum weight base $B$ of $\mathcal{M}$ that the data structure maintains has the property that $S \subseteq B$ at all times, and $B \setminus S$ is a maximum weight base in $\mathcal{M}/S$. This follows from the observation that, if an element $e$ is in the maximum weight base $B$ and the only changes to the weights are such that the weight of $e$ remains unchanged and the weights of elements other than $e$ are decreased, then $e$ remains in the new maximum weight base.

**The discrete Greedy algorithm.** The algorithm (Algorithm 2) is based on the random residual Greedy algorithm of [4]. The latter algorithm constructs a solution $S$ over $k$ iterations.

**Algorithm 2** LazySamplingGreedy($f, \mathcal{M}, \epsilon$).

```
1: $M = \Theta(f(\mathrm{OPT}))$, $c = \Theta(1/\epsilon)$, $N = 2\ln(k/\epsilon)/\epsilon$
2: $\langle\langle$ maintain cached (rounded) marginal values $\rangle\rangle$
3: For each $e \in V$, let $w(e) = (1-\epsilon)^N M$ if $f(\{e\}) \le (1-\epsilon)^N M$ and $w(e) = (1-\epsilon)^{j-1} M$ if
   $f(\{e\}) \in ((1-\epsilon)^j M, (1-\epsilon)^{j-1} M]$
4: $\langle\langle$ maintain a base $B$ of maximum $w(\cdot)$ value in a data structure that
   supports the UpdateBase operation $\rangle\rangle$
5: $B = \mathrm{argmax}_{S \in \mathcal{I}} \sum_{e \in S} w(e)$
6: $\langle\langle$ maintain a partition of $B$ into buckets $\rangle\rangle$
7: $B^{(j)} = \{e \in B : w(e) = (1-\epsilon)^{j-1} M\}$ for each $j \in [N]$
8: $W = \sum_{e \in B} w(e)$
9: $\langle\langle$ main loop $\rangle\rangle$
10: $S = \emptyset$
11: for $t = 1, 2, \ldots, k$ do
12:     Call RefreshValues
13:     if $W \le 4cM$ then
14:         return $S$
15:     end if
16:     Sample an element $e$ uniformly at random from $B$
17:     $S \leftarrow S \cup \{e\}$
18:     Remove $e$ from the buckets of $B$ for refreshing purpose so that $w(e)$ is now fixed
19: end for
1: procedure RefreshValues                    $\langle\langle$ Spot check and update values $\rangle\rangle$
2:     for $j = 1$ to $N$ do
3:         $T = 0$
4:         while $T < 4\log_2 n$ do
5:             if $B^{(j)}$ is empty then
6:                 Exit the while loop and continue to iteration $j+1$
7:             end if
8:             Sample $e$ uniformly at random from $B^{(j)}$
9:             Let $v(e) = f(S \cup \{e\}) - f(S)$ be the current marginal value of $e$
10:            if $v(e) < (1-\epsilon)^j M$ then
11:                $T = 0$
12:                UpdateBase($e, j, v(e)$)
13:            else
14:                $T \leftarrow T + 1$
15:            end if
16:        end while
17:    end for
18: end procedure
```

In each iteration, the algorithm assigns a linear weight to each element that is equal to the marginal gain $f(S \cup \{e\}) - f(S)$ on top of the current solution, and it finds a maximum weight base $B$ in $\mathcal{M}/S$. The algorithm then samples an element of $B$ uniformly at random and adds it to the solution. As discussed in Section 1.2, the key difficulty for obtaining a fast running time is maintaining the maximum weight base. Our algorithm uses the following approach for maintaining an approximate maximum weight base. The algorithm maintains the marginal value of each element (rounded to the next highest power of $(1-\epsilon)$), and it updates it in a lazy manner; at every point, $w(e)$ denotes the cached (rounded) marginal value of the element, and it may be stale.

The algorithm maintains the base $B$ using the data structure discussed above that supports the UPDATEBASE operation. Additionally, the elements of $B \setminus S$ are stored into buckets corresponding to geometrically decreasing marginal values. More precisely, there are $N = O(\log(k/\epsilon)/\epsilon)$ buckets $B^{(1)}, B^{(2)}, \ldots, B^{(N)}$. The $j$-th bucket $B^{(j)}$ contains all of the elements of $B$ with marginal values in the range $((1-\epsilon)^j M, (1-\epsilon)^{j-1} M]$, where $M$ is a value such that $f(\mathrm{OPT}) \leq M \leq O(1) f(\mathrm{OPT})$ (we assume that the algorithm knows such a value $M$, as it can be obtained in nearly-linear time, see e.g. Lemma 3.2 in [5]). The remaining elements of $B$ that do not appear in any of the $N$ buckets have marginal values at most $(1-\epsilon)^N M$; these elements have negligible total marginal gain, and they can be safely ignored.

In order to achieve a fast running time, after each Greedy step, the algorithm uses sampling to (partially) update the base $B$, the cached marginal values, and the buckets. This is achieved by the procedure REFRESHVALUES, which works as follows. REFRESHVALUES considers each of the buckets in turn. The algorithm updates the bucket $B^{(j)}$ by spot checking $O(\log n)$ elements sampled uniformly at random from the bucket. For each sampled element $e$, the algorithm computes its current marginal value and, if it has decreased below the range of its bucket, it moves the element to the correct buckets and call UPDATEBASE to maintain the invariant that $B$ is a maximum weight base.

When the algorithm finds an element whose bucket has changed, it resets to 0 the count for the number of samples taken from the bucket. Thus the algorithm keeps sampling from the bucket until $\Theta(\log n)$ consecutive sampled elements do not change their bucket. The sampling step ensures that, with high probability, in each bucket at least half of the elements are in the correct bucket. (We remark that, instead of resetting the sample count to 0, it suffices to decrease the count by 1, i.e., the count is the total number of samples whose bucket was correct minus the number of samples whose bucket was incorrect. The algorithm then stops when this tally reaches $\Theta(\log n)$. This leads to an improvement in the running time, but we omit it in favor of a simpler analysis.)

After running REFRESHVALUES, the algorithm samples an element $e$ uniformly at random from $B \setminus S$ and adds it to $S$. The algorithm then removes $e$ from the buckets; this ensures that the weight of $e$ will remain unchanged for the remainder of the algorithm.

## 2.1 Analysis of the approximation guarantee

Here we show that Algorithm 1 achieves a $1 - 1/e - \epsilon$ approximation. We first analyze the LAZYSAMPLINGGREEDY algorithm. We start with some convenient definitions. Consider some point in the execution of the LAZYSAMPLINGGREEDY algorithm. Consider a bucket $B^{(j)}$. At this point, each element $e \in B^{(j)}$ is in the correct bucket iff its current marginal value $f(S \cup \{e\}) - f(S)$ lies in the interval $((1-\epsilon)^j M, (1-\epsilon)^{j-1} M]$ (its cached marginal value $w(e)$ lies in that interval, but it might be stale). We say that the bucket $B^{(j)}$ is *good* if at least half of the elements in $B^{(j)}$ are in the correct bucket, and we say that the bucket is *bad* otherwise.

The following lemma shows that, with high probability over the random choices of REFRESHVALUES, each run of REFRESHVALUES ensures that every bucket $B^{(j)}$ with $j \in [N]$ is good.

▶ **Lemma 8.** *Consider an iteration in which* LAZYSAMPLINGGREEDY *calls* REFRESHVALUES. *When* REFRESHVALUES *terminates, the probability that the buckets $\{B^{(j)} : j \in [N]\}$ are all good is at least $1 - 1/n^2$.*

**Proof.** We will show that the probability that a given bucket is bad is at most $5\log n/n^3$; the claim then follows by the union bound, since there are $N \le n/(5\log n)$ buckets. Consider a bucket $B^{(j)}$, where $j \in [N]$, and suppose that the bucket is bad at the end of RefreshValues. We analyze the probability the bucket is bad because the algorithm runs until iteration $t$, which is the last time the algorithm finds an element in $B^{(j)}$ in the wrong bucket, and for $4\log n$ iterations after $t$, it always find elements in the right bucket even though only $1/2$ of $B^{(j)}$ are in the right bucket. Since at most half of the elements of $B^{(j)}$ are in the correct bucket and the samples are independent, this event happens with probability at most $(1/2)^{4\log_2 n} = 1/n^4$. By the union bound over all choices of $t = 1, 2, \ldots, 5n\log n$, the failure probability for bucket $B^{(j)}$ is at most $5\log n/n^3$. ◀

Since LazySamplingGreedy performs at most $k \le n$ iterations, it follows by the union bound that all of the buckets $\{B^{(j)} : j \in [N]\}$ are all good throughout the algorithm with probability at least $1 - 1/n$. For the remainder of the analysis, we condition on this event. Additionally, we fix an event specifying the random choices made by RefreshValues and we implicitly condition all probabilities and expectations on this event.

Let us now show that $B$ is a suitable approximation for the maximum weight base in $\mathcal{M}/S$ with weights given by the current marginal values $f(S \cup \{e\}) - f(S)$.

▶ **Lemma 9.** *Suppose that every bucket of $B$ is good throughout the algorithm. Let $v(e) = f(S \cup \{e\}) - f(S)$ denote the current marginal values. We have*
**(1)** *$w(S') \ge v(S')$ for every $S' \subseteq V$;*
**(2)** *$w(B) \ge w(S')$ for every $S' \subseteq V$;*
**(3)** *$v(B) \ge \frac{1-\epsilon}{2} \cdot w(B) - \frac{\epsilon^2}{k} \cdot M$.*

**Proof.** The first property follows from the fact that, by submodularity, the weights $w(\cdot)$ are upper bounds on the marginal values.

The second property follows from the fact that the algorithm maintains the invariant that $B$ is the maximum weight base in $\mathcal{M}/S$ with respect to the weights $w(\cdot)$.

Let us now show the third property. Consider the following partition of $B$ into sets $B_1$, $B_2$, and $B_3$, where: $B_1$ is the set of all elements $e \in B$ such that $e$ is in one of the buckets $\{B^{(j)} : j \in [N]\}$ and moreover $e$ is in the correct bucket (note that $(1-\epsilon)w(e) \le v(e) \le w(e)$ for every $e \in B_1$); $B_2$ is the set of all elements $e \in B$ such that $e$ is in one of the buckets $\{B^{(j)} : j \in [N]\}$ but $e$ is not in the correct bucket (i.e., $(1-\epsilon)^N M < w(e)$ but $v(e) < (1-\epsilon)w(e)$); $B_3$ is the set of all elements $e \in B$ such that $w(e) < (1-\epsilon)^N M \le (\epsilon/k)^2 M$.

Since $|B_3| \le k$, we have $w(B_3) \le |B_3| \cdot \left(\frac{\epsilon}{k}\right)^2 M \le \frac{\epsilon^2}{k} M$.

Since all of the buckets are good, it follows that the total $w(\cdot)$ weight of the elements that are in the correct bucket is at least $\frac{1}{2} \cdot w(B \setminus B_3)$. Indeed, we have

$$w(B_1) = \sum_{j=1}^{N} w(B_1 \cap B^{(j)}) = \sum_{j=1}^{N} (1-\epsilon)^{j-1} M |B_1 \cap B^{(j)}| \ge \sum_{j=1}^{N} (1-\epsilon)^{j-1} M \frac{|B^{(j)}|}{2} = \frac{w(B \setminus B_3)}{2}.$$

Finally, since $v(e) \ge (1-\epsilon)w(e)$ for every $e \in B_1$, we have

$$v(B) \ge v(B_1) \ge (1-\epsilon)w(B_1) \ge \frac{1-\epsilon}{2} w(B \setminus B_3) \ge \frac{1-\epsilon}{2} w(B) - \frac{\epsilon^2}{k} M.$$ ◀

Now we turn to the analysis of the main for loop of LazySamplingGreedy (lines 11–19). Let $Z$ be a random variable equal to the number of iterations where the algorithm executes line 17. We define sets $\{S_t : t \in \{0, 1, \ldots, k\}\}$ and $\{\mathrm{OPT}_t : t \in \{0, 1, \ldots, k\}\}$ as follows. Let $S_0 = \emptyset$ and $\mathrm{OPT}_0 = \mathrm{OPT}$. Consider an iteration $t \le Z$ and suppose that

$S_{t-1}$ and $\mathrm{OPT}_{t-1}$ have already been defined and they satisfy $S_{t-1} \cup \mathrm{OPT}_{t-1} \in \mathcal{I}$ and $|S_{t-1}| + |\mathrm{OPT}_{t-1}| = k$. Consider a bijection $\pi : B \to \mathrm{OPT}_{t-1}$ so that $\mathrm{OPT}_{t-1} \setminus \{\pi(e)\} \cup \{e\}$ is a base of $\mathcal{M}/S_{t-1}$ for all $e \in B$ (such a bijection always exists, see e.g., Corollary 39.12a in [22]). Let $e_t$ be the element sampled on line 17 and $o_t = \pi(e_t)$. We define $S_t = S_{t-1} \cup \{e_t\}$ and $\mathrm{OPT}_t = \mathrm{OPT}_{t-1} \setminus \{o_t\}$. Note that $S_t \cup \mathrm{OPT}_t \in \mathcal{I}$. For each $t > Z$, we define $S_t = S_Z$ and $\mathrm{OPT}_t = \mathrm{OPT}_Z$.

In each iteration $t$, the gain in the Greedy solution value is $f(S_t) - f(S_{t-1})$, and the loss in the optimal solution value is $f(\mathrm{OPT}_{t-1}) - f(\mathrm{OPT}_t)$ (when we add an element to $S_{t-1}$, we remove an element from $\mathrm{OPT}_{t-1}$ so that $S_t \cup \mathrm{OPT}_t$ remains a feasible solution). The following lemma relates the two values in expectation.

▶ **Lemma 10.** *For every $t \in [k]$, if all of the buckets $B^{(j)}$ are good, we have* $\mathbb{E}[f(S_t) - f(S_{t-1})] \geq c \cdot \mathbb{E}[f(\mathrm{OPT}_{t-1}) - f(\mathrm{OPT}_t)]$.

**Proof.** Consider $t \in [k]$. Recall that $Z$ is the number of iterations where the algorithm executes line 17. If $t > Z$, the inequality is trivially satisfied, since both expectations are equal to 0. Therefore we may assume that $t \leq Z$ and thus $S_t = S_{t-1} \cup \{e_t\}$ and $\mathrm{OPT}_t = \mathrm{OPT}_{t-1} \setminus \{o_t\}$.

Let us now fix an event $R_{t-1}$ specifying the random choices for the first $t-1$ iterations, i.e., the random elements $e_1, \dots, e_{t-1}$ and $o_1, \dots, o_{t-1}$. In the following, all the probabilities and expectations are implicitly conditioned on $R_{t-1}$. Note that, once $R_{t-1}$ is fixed, $S_{t-1}$ and $\mathrm{OPT}_{t-1}$ are deterministic.

Let us first lower bound $\mathbb{E}[f(S_{t-1} \cup \{e_t\}) - f(S_{t-1})]$. Let $w_t$, $B_t$, and $W_t$ denote $w$, $B$, and $W$ right after executing RefreshValues in iteration $t$. Note that, since $R_{t-1}$ and the random choices of RefreshValues are fixed, $w_t$, $B_t$, and $W_t$ are deterministic.

Recall that all of the buckets of $B_t$ are good, i.e., at least half of the elements of $B_t^{(j)}$ are in the correct bucket, for every $j \in [N]$. Let $B'_t$ be the subset of $B_t$ consisting of all of the elements that are in the correct bucket, and let $B''_t$ be the subset of $B_t$ consisting of all of the elements that are not in any bucket.

For every $e \in B'_t$, we have $f(S_{t-1} \cup \{e\}) - f(S_{t-1}) \geq (1-\epsilon) w_t(e)$. For every $e \in B''_t$, we have $f(S_{t-1} \cup \{e\}) - f(S_{t-1}) \leq w_t(e) = (1-\epsilon)^N M \leq (\epsilon/k)^2 M$, and therefore $w_t(B''_t) \leq \frac{\epsilon^2}{k} M$. Since all of the buckets are good and $W_t > 4cM$ (the algorithm did not terminate on line 14), we have

$$\sum_{e \in B'_t} w_t(e) = \sum_{j=1}^{N} w_t(B'_t \cap B_t^{(j)}) = \sum_{j=1}^{N} (1-\epsilon)^{j-1} M |B'_t \cap B_t^{(j)}| \geq \sum_{j=1}^{N} (1-\epsilon)^{j-1} M \frac{|B_t^{(j)}|}{2}$$
$$= \frac{w_t(B_t \setminus B''_t)}{2} \geq \frac{W_t}{2} - \frac{\epsilon^2}{2k} M \geq \left(2c - \frac{\epsilon^2}{2k}\right) M \geq \left(2c - \frac{\epsilon^2}{2k}\right) f(\mathrm{OPT}).$$

By combining these observations, we obtain

$$\mathbb{E}[f(S_{t-1} \cup \{e_t\}) - f(S_{t-1})] \geq \mathbb{E}[f(S_{t-1} \cup \{e_t\}) - f(S_{t-1}) | e_t \in B'_t] \Pr[e_t \in B'_t]$$
$$= \mathbb{E}[f(S_{t-1} \cup \{e_t\}) - f(S_{t-1}) | e_t \in B'_t] \cdot \frac{|B'_t|}{|B_t|} \geq (1-\epsilon) \mathbb{E}[w_t(e_t) | e_t \in B'_t] \cdot \frac{|B'_t|}{|B_t|}$$
$$= (1-\epsilon) w_t(B'_t) \cdot \frac{1}{|B'_t|} \cdot \frac{|B'_t|}{|B_t|} \geq \frac{(1-\epsilon)\left(2c - \frac{\epsilon^2}{2k}\right)}{|B_t|} f(\mathrm{OPT}) \geq \frac{c}{|B_t|} f(\mathrm{OPT})$$

Let us now upper bound $\mathbb{E}[f(\mathrm{OPT}_{t-1}) - f(\mathrm{OPT}_t)]$. Recall that $e_t$ is chosen randomly from

$B$ and thus, $o_t$ is chosen uniformly at random from $\mathrm{OPT}_{t-1}$ (since $\pi$ is a bijection). Hence

$$\mathbb{E}[f(\mathrm{OPT}_{t-1}) - f(\mathrm{OPT}_{t-1} \setminus \{o_t\})] = \sum_{o \in \mathrm{OPT}_{t-1}} (f(\mathrm{OPT}_{t-1}) - f(\mathrm{OPT}_{t-1} \setminus \{o\})) \cdot \frac{1}{|\mathrm{OPT}_{t-1}|}$$

By submodularity, we have

$$f(\mathrm{OPT}_{t-1}) \geq \sum_{j=1}^{m} (f(\mathrm{OPT}_{t-1}) - f(\mathrm{OPT}_{t-1} \setminus \{o_j\}).$$

Therefore

$$\mathbb{E}[f(\mathrm{OPT}_{t-1}) - f(\mathrm{OPT}_{t-1} \setminus \{o_t\})] \leq \frac{f(\mathrm{OPT}_{t-1})}{|\mathrm{OPT}_{t-1}|} \leq \frac{f(\mathrm{OPT})}{|\mathrm{OPT}_{t-1}|}.$$

To recap, we have shown that:

$$\mathbb{E}[f(S_{t-1} \cup \{e_t\}) - f(S_{t-1})] \geq \frac{c \cdot f(\mathrm{OPT})}{|B_t|},$$
$$\mathbb{E}[f(\mathrm{OPT}_{t-1}) - f(\mathrm{OPT}_{t-1} \setminus \{o_t\})] \leq \frac{f(\mathrm{OPT})}{|\mathrm{OPT}_{t-1}|}.$$

Since $|B_t| = |\mathrm{OPT}_{t-1}|$, we have

$$\mathbb{E}[f(S_{t-1} \cup \{e_t\}) - f(S_{t-1})] \geq c \cdot \mathbb{E}[f(\mathrm{OPT}_{t-1}) - f(\mathrm{OPT}_{t-1} \setminus \{o_t\})].$$

Since the above inequality holds conditioned on every given event $R_{t-1}$, it holds unconditionally, and the lemma follows. ◀

The following lemma follows from Lemmas 9 and 10.

▶ **Lemma 11.** *If all of the buckets $B^{(j)}$ are good, the* LazySamplingGreedy *algorithm (Algorithm 2) returns a set $S \in \mathcal{I}$ with the following properties.*

**(1)** $\max_{S' \colon S' \cup S \in \mathcal{I}} \sum_{e \in S'} f_S(e) \leq 4cM = O(1/\epsilon) f(\mathrm{OPT})$.

**(2)** *There is a random subset $\mathrm{OPT}' \subseteq \mathrm{OPT}$ depending on $S$ with the following properties: $S \cup \mathrm{OPT}' \in \mathcal{I}$ and $\mathbb{E}[f(\mathrm{OPT}')] \geq f(\mathrm{OPT}) - \frac{1}{c} \cdot \mathbb{E}[f(S)] \geq \left(1 - \frac{1}{c}\right) f(\mathrm{OPT})$.*

By combining Lemmas 7 and 11, we obtain:

▶ **Lemma 12.** *The* ContinuousMatroid *algorithm (Algorithm 1) returns a solution $\mathbf{1}_S \vee x \in P(\mathcal{M})$ such that $F(\mathbf{1}_S \vee x) \geq (1 - 1/e - O(\epsilon)) f(\mathrm{OPT})$ with constant probability.*

**Proof.** Note that, in order to apply Lemma 7, we need the following condition to hold: $\max_{S' \colon S' \cup S \in \mathcal{I}} \sum_{e \in S'} f_S(e) \leq c' f_S(\mathrm{OPT}'')$, where $\mathrm{OPT}'' \in \mathrm{argmax}_{S' \colon S' \cup S \in \mathcal{I}} f_S(S')$.

Using Lemma 11, we can show that the above condition holds with constant probability as follows. Let $\mathrm{OPT}'$ be the set guaranteed by Lemma 11. We have $f_S(\mathrm{OPT}') \leq f_S(\mathrm{OPT}'')$ and $f(S \cup \mathrm{OPT}') \geq f(\mathrm{OPT}')$. Therefore $f_S(\mathrm{OPT}'') \geq f_S(\mathrm{OPT}') \geq f(\mathrm{OPT}') - f(S)$.

By Lemma 11, we have $\mathbb{E}[f(\mathrm{OPT}) - f(\mathrm{OPT}')] \leq f(\mathrm{OPT})/c$. Therefore, by the Markov inequality, with probability at least $2/3$, we have $f(\mathrm{OPT}) - f(\mathrm{OPT}') \leq 3f(\mathrm{OPT})/c$. Consider two cases. First, if $f(S) \geq (1-1/e)f(\mathrm{OPT})$ then the algorithm can simply return $S$. Second, if $f(S) < (1 - 1/e)f(\mathrm{OPT})$ then $f_S(\mathrm{OPT}'') \geq f(\mathrm{OPT}') - f(S) \geq (1/e - 3/c)f(\mathrm{OPT})$. Therefore, $\max_{S' \colon S' \cup S \in \mathcal{I}} \sum_{e \in S'} f_S(e) \leq O(c f_S(\mathrm{OPT}'')) \leq c' f_S(\mathrm{OPT}'')$. Thus the conditions of Lemma 7 are satisfied and thus the continuous Greedy algorithm returns a solution $x \in P(\mathcal{M}/S)$ such that

$$F(\mathbf{1}_S \vee x) - f(S) \geq \left(1 - \frac{1}{e} - \epsilon\right) (f(\mathrm{OPT}') - f(S))$$
$$\geq \left(1 - \frac{1}{e} - \epsilon\right) \left(1 - \frac{3}{c}\right) f(\mathrm{OPT}) - f(S) \geq \left(1 - \frac{1}{e} - 2\epsilon\right) f(\mathrm{OPT}) - f(S).$$ ◀

## References

1 Alexander Ageev and Maxim Sviridenko. Pipage Rounding: A New Method of Constructing Algorithms with Proven Performance Guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

2 Yossi Azar and Iftah Gamzu. Efficient Submodular Function Maximization under Linear Packing Constraints. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2012.

3 Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014.

4 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular Maximization with Cardinality Constraints. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014.

5 Niv Buchbinder, Moran Feldman, and Roy Schwartz. Comparing Apples and Oranges: Query Trade-off in Submodular Maximization. *Math. Oper. Res.*, 42(2):308–329, 2017.

6 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a Submodular Set Function Subject to a Matroid Constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

7 Chandra Chekuri, T. S. Jayram, and Jan Vondrák. On Multiplicative Weight Updates for Concave and Submodular Function Maximization. In *Conference on Innovations in Theoretical Computer Science (ITCS)*, 2015. doi:10.1145/2688073.2688086.

8 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent Randomized Rounding via Exchange Properties of Combinatorial Structures. In *IEEE Foundations of Computer Science (FOCS)*, pages 575–584. IEEE Computer Society, 2010.

9 Shaddin Dughmi, Tim Roughgarden, and Mukund Sundararajan. Revenue Submodularity. *Theory of Computing*, 8(1):95–119, 2012.

10 Uriel Feige. A threshold of ln n for approximating set cover. *jacm*, 45:634–652, 1998.

11 Yuval Filmus and Justin Ward. Monotone Submodular Maximization over a Matroid via Non-Oblivious Local Search. *SIAM Journal on Computing*, 43(2):514–542, 2014.

12 M L Fisher, G L Nemhauser, and L A Wolsey. An analysis of approximations for maximizing submodular set functions—II. *Mathematical Programming Studies*, 8:73–87, 1978.

13 Bernard A Galler and Michael J Fisher. An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303, 1964.

14 Ryan Gomes and Andreas Krause. Budgeted Nonparametric Learning from Data Streams. In *International Conference on Machine Learning (ICML)*, pages 391–398, 2010.

15 Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.

16 David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 137–146, 2003.

17 Andreas Krause, Ajit Paul Singh, and Carlos Guestrin. Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. *Journal of Machine Learning Research*, 9:235–284, 2008.

18 Hui Lin and Jeff A. Bilmes. Multi-document Summarization via Budgeted Maximization of Submodular Functions. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics*, pages 912–920, 2010.

19 Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier Than Lazy Greedy. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

20 G L Nemhauser and L A Wolsey. Best Algorithms for Approximating the Maximum of a Submodular Set Function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

21 G L Nemhauser, L A Wolsey, and M L Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978.

22 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

23 Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM (JACM)*, 22(2):215–225, 1975.

24 Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *ACM Symposium on Theory of Computing (STOC)*, 2008.