Submodular Maximization with Matroid and Packing Constraints in Parallel

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

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

Adrian Vladu
Department of Computer Science, Boston University
Boston, MA, USA
avladu@mit.edu

ABSTRACT

We consider the problem of maximizing the multilinear extension of a submodular function subject a single matroid constraint or multiple packing constraints with a small number of adaptive rounds of evaluation queries.

We obtain the first algorithms with low adaptivity for submodular maximization with a matroid constraint. Our algorithms achieve a $1 - 1/e - \epsilon$ approximation for monotone functions and a $1/e - \epsilon$ approximation for non-monotone functions, which nearly matches the best guarantees known in the fully adaptive setting. The number of rounds of adaptivity is $O(\log^2 n/\epsilon^3)$, which is an exponential speedup over the existing algorithms.

We obtain the first parallel algorithm for non-monotone submodular maximization subject to packing constraints. Our algorithm achieves a $1/e - \epsilon$ approximation using $O(\log(n/\epsilon)\log(1/\epsilon)\log(n+m)/\epsilon^2)$ parallel rounds, which is again an exponential speedup in parallel time over the existing algorithms. For monotone functions, we obtain a $1-1/e-\epsilon$ approximation in $O(\log(n/\epsilon)\log m/\epsilon^2)$ parallel rounds. The number of parallel rounds of our algorithm matches that of the state of the art algorithm for solving packing LPs with a linear objective (Mahoney et al., 2016).

Our results apply more generally to the problem of maximizing a diminishing returns submodular (DR-submodular) function.

CCS CONCEPTS

• **Theory of computation → Approximation algorithms analysis**; **Continuous optimization**; **Parallel algorithms**; *Packing and covering problems*; Discrete optimization.

KEYWORDS

DR-submodular maximization, packing, matroid, parallel complexity

ACM Reference Format:
Alina Ene, Huy L. Nguyễn, and Adrian Vladu. 2019. Submodular Maximization with Matroid and Packing Constraints in Parallel. In *Proceedings of the 51st Annual ACM SIGACT Symposium on the Theory of Computing (STOC '19), June 23–26, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3313276.3316389

STOC '19, June 23–26, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6705-9/19/06…$15.00
https://doi.org/10.1145/3313276.3316389

1 INTRODUCTION

A set function f on a finite ground set V is submodular if it satisfies the following *diminishing returns* property: $f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B)$ for all sets $A \subseteq B$ and all elements $v \in V \setminus B$. The general problem of optimizing a submodular function subject to constraints captures many problems of interest both in theory and in practice, including maximum coverage, social welfare maximization, influence maximization in social networks, sensor placement, maximum cut, minimum cut, and facility location. Submodular optimization problems have received considerable attention over the years, leading to the development of a rich theory and applications in a wide-range of areas such as machine learning, computer vision, data mining, and economics. At a high level, these developments have established that diminishing returns often implies tractability: submodular functions can be minimized in polynomial time and they can be approximately maximized subject to a wide range of constraints.

More recently, the diminishing returns property has been generalized and studied in continuous domains [2, 8–10, 28, 29]. Most of these works study the following *continuous diminishing returns submodular* (DR-submodular) property: a differentiable function is DR-submodular if and only if the gradient is monotone decreasing (if $\vec{x} \leq \vec{y}$ coordinate-wise, $\nabla f(\vec{x}) \geq \nabla f(\vec{y})$), and a twice-differentiable function is DR-submodular if and only if all the entries of the Hessian are non-positive ($\frac{\partial^2 f(\vec{x})}{\partial x_i \partial x_j} \leq 0$ for all $i, j \in [n]$). DR-submodular optimization bridges continuous and discrete optimization. Indeed, the multilinear extension of a submodular set function is a DR-submodular function and DR-submodular maximization was studied in the context of submodular maximization starting with the influential work of Calinescu *et al.* [12]. The multilinear relaxation framework is a very general and powerful approach for submodular maximization and it has led to the current best approximation algorithms for a wide variety of constraints including cardinality constraints, knapsack constraints, matroid constraints, etc. Recent work has shown that DR-submodular optimization problems have applications beyond submodular maximization [8–10, 29].

The problem of maximizing a DR-submodular function subject to a convex constraint is a notable example of a *non-convex* optimization problem that can be solved with *provable* approximation guarantees. The continuous Greedy algorithm [30] developed in the context of the multilinear relaxation framework applies more generally to maximizing DR-submodular functions that are monotone

increasing (if $\vec{x} \leq \vec{y}$ coordinate-wise then $f(\vec{x}) \leq f(\vec{y})$). Chekuri *et al.* [13] developed algorithms for both monotone and non-monotone DR-submodular maximization subject to packing constraints that are based on the continuous Greedy and multiplicative weights update framework.

A significant drawback of these algorithms is that they are inherently sequential and adaptive. Recent lines of work have focused on addressing these shortcomings and understanding the trade-offs between approximation guarantee, parallelization, and adaptivity. These efforts have led to the development of distributed algorithms for submodular maximization in parallel models of computation such as MapReduce [6, 7, 17, 21, 24–26]. These works parallelize the Greedy algorithm and its variants and achieve various tradeoffs between the approximation guarantee and the number of rounds of MapReduce computation and other resources, such as memory and communication. The algorithms developed in these works run sequential Greedy algorithms on each of the machines and thus their decisions remain highly adaptive. Starting with the work of Balkanski and Singer [5], there have been very recent efforts to understand the tradeoff between approximation guarantee and adaptivity for submodular maximization [3–5, 14, 15, 18]. The *adaptivity* of an algorithm is the number of sequential rounds of queries it makes to the evaluation oracle of the function, where in every round the algorithm is allowed to make polynomially-many parallel queries. Most of these works have focused on monotone submodular maximization with a cardinality constraint, leading to a nearly-optimal $1-1/e-\epsilon$ approximation and nearly-optimal $O(\log n/\epsilon^2)$ adaptivity. Chekuri and Quanrud [14] consider the more general setting of multiple packing constraints, which capture several constraints of interest, including cardinality, knapsack, partition and laminar matroids, matchings, and their intersection. They give an algorithm for monotone DR-submodular maximization with packing constraints that is based on the continuous Greedy and multiplicative weights update frameworks [13]. The algorithm achieves a $1-1/e-\epsilon$ approximation using $O(\log(n/\epsilon)\log^2 m/\epsilon^4)$ rounds of adaptivity, where m is the number of packing constraints.

This line of works is related to parallel algorithms for optimizing linear objectives subject to constraints such as packing linear programs. This problem was originally studied in the pioneering work of Luby and Nisan [22] with $O(\log^2 n/\epsilon^4)$ rounds of parallel computation. This bound was the best known for decades before the recent improvement to $O(\log^2 n/\epsilon^3)$ by [1] and then to the state of the art $O(\log^2 n/\epsilon^2)$ by [23].

Despite the significant progress, there are several significant challenges that remain in submodular maximization with low adaptivity. First, a lot these algorithms are ad-hoc and significantly exploiting the simplicity of the cardinality constraint. Second, most of these algorithms can only handle the monotone objective function and it is not clear how to get close to $1/e$ approximation, even for a single cardinality constraint. Indeed, Chekuri and Quanrud [14] identify the non-monotone case as a significant open problem in this area.

1.1 Our Techniques and Contributions

In this work, we address both of the above challenges, and simultaneously (1) design a novel interface between algorithms for linear objective and algorithms for submodular objective, and (2) develop generic techniques for handling non-monotone objectives. Our new techniques lead to new algorithms with nearly optimal approximations for multiple packing constraints as well as a single matroid constraint for both monotone and non-monotone submodular objectives in poly-logarithmic number of rounds of adaptivity. Note that for the case of a matroid constraint, no algorithm better than Greedy was known and our algorithms achieve an *exponential* improvement in adaptivity. Before our work, it was also not known how to get close to $1/e$ approximation for non-monotone objectives in a sublinear number of rounds, even for a single cardinality constraint.

We now describe at a high level some of the main difficulties in parallelizing the existing algorithms and our approach for overcoming them. In previous continuous Greedy algorithms for a matroid constraint, the algorithm repeatedly computes a maximum weight base with respect to the current gradient and adds it to the solution. The gradient is used as a proxy for the change in the objective value when the solution changes. A problem with this approach is that the gradient changes quickly when we update a lot of coordinates and, as a result, the algorithm can only take small steps and it needs a linear number of adaptive rounds. Another approach is to change one coordinate at a time and update the gradient every time, which leads to a faster algorithm but still a linear number of adaptive rounds (the number of coordinate updates could be linear).

In contrast, our algorithms use *multiplicative updates* instead of additive updates and the *gradient at (an upper bound of) the future point* instead of the current point. Using the gradient at the future point ensures that we never overestimate the gain (due to the diminishing returns of the objective), which was the problem that led to the small steps in the previous works. The multiplicative updates allow us to have a safe guess for (an upper bound of) the future point without underestimating the gain too much. These techniques lead to an algorithm for monotone objectives with nearly optimal approximation and poly-logarithmic number of rounds of adaptivity. For the non-monotone case, our algorithm naturally combines with the measured continuous Greedy algorithm to obtain a $1/e$ approximation. It should be emphasized that this combination is enabled by our above technique for estimating the gain correctly even for large steps and when there are negative gains. Previous works use the gradient at the current solution, which overestimates the gain due to diminishing return and it can be detrimental when the overestimation changes negative values to positive values. This issue leads to sophisticated analyses even for a monotone objective, let alone the non-monotone case. Indeed, [14] identified this issue as a major obstacle preventing their techniques from being applicable to non-monotone objectives.

For the packing constraints, the previous work [14] uses the approach of Young [31] for packing LPs and the resulting proof is fairly complex because of the overestimation of the gain from the gradient. Their algorithm needs to find other ways to make progress (via filtering) when the overestimation is beyond the tolerable error. Instead, our technique above allows for accurate estimation of the gain even when there are negative gains, which allows us to use state of the art techniques for packing LPs [23] and retain much of the simplicity and elegance of their algorithm and proof for the linear objectives. We note that our analysis is substantially more involved than in the linear setting, since the linear approximation

to the submodular function given by the gradient is changing over time.

Our algorithm and analysis for packing constraints are a significant departure from previous work such as [14]. The algorithm of [14] deals with the changing objective by dividing the execution of the algorithm into phases where, within each phase, the objective value increases by ϵ times the optimal value. This division makes the analysis easier. For instance, the Greedy algorithm would like to pick coordinates whose gain is proportional to the difference between the optimal value and the current solution value. Within each phase, up to an $1+\epsilon$ factor, this threshold is the same. The total saturation of the constraints also behaves similarly. The fact that, up to an $1+\epsilon$ approximation, all relevant quantities are constant is extremely useful in this context because one can adapt the analysis from the case of a linear objective. However, this partition can lead to a suboptimal number of iterations: there are $\Omega(1/\epsilon)$ phases and the number of iterations might be suboptimal by a $\Omega(1/\epsilon)$ factor. In contrast, our algorithm does not use phases and our analysis has a global argument for bounding the number of iterations. A global argument exists for the linear case, but here we need a much more general argument to handle the non-linear objective. The resulting argument is intricate and requires the division of the iterations into only two parts, instead of $\Omega(1/\epsilon)$ phases.

Our algorithms only rely on DR-submodularity and they further draw an important distinction between convex optimization and (non-convex) DR-submodular optimization: Nemirovski [27] showed that there are unconstrained minimization problems with a non-smooth convex objective for which any parallel algorithm requires $\Omega((n/\log n)^{1/3}\log(1/\epsilon))$ rounds of adaptivity to construct an ϵ-optimal solution, whereas the adaptive complexity of DR-submodular maximization is *exponentially smaller* in the dimension.

1.2 Our Results

Our contributions for a matroid constraint are the following.

Theorem 1. *For every $\epsilon > 0$, there is an algorithm for maximizing a DR-submodular function $f : [0,1]^n \to \mathbb{R}_+$ subject to the constraint $\vec{x} \in \mathbf{P}$, where $\mathbf{P}$ is a matroid polytope, with the following guarantees:*

- *The algorithm is deterministic if provided oracle access for evaluating f and its gradient ∇f;*
- *The algorithm achieves an approximation guarantee of $1 - 1/e - \epsilon$ for monotone functions and $1/e - \epsilon$ for non-monotone functions;*
- *The number of rounds of adaptivity and evaluations of f and ∇f are $O\left(\frac{\log^2 n}{\epsilon^3}\right)$.*

The guarantee on the number of rounds of adaptivity is under the assumption that, for every vector $\vec{x}$, the entries of the gradient $\nabla f(\vec{x})$ are at most $\mathrm{poly}(n/\epsilon) f(\vec{x}^*)$, where $f(\vec{x}^*)$ is the optimal solution value. This assumption is satisfied when f is the multilinear extension of a submodular function, since the gradient entries are upper bounded by the singleton values.

Our algorithm is the first low-adaptivity algorithm for submodular maximization with a matroid constraint, and it achieves an exponential speedup in the number of adaptive rounds over the existing algorithms with only an arbitrarily small loss in the approximation guarantee. We note that the algorithm is not a parallel algorithm in the **NC** sense, since we are working with a general polymatroid constraint and checking feasibility involves minimizing the so-called border function of the polymatroid, which is a general submodular function minimization problem (see e.g., Chapter 5 in [20]).

Our contributions for packing constraints are the following.

Theorem 2. *For every $\epsilon > 0$, there is an algorithm for maximizing a monotone DR-submodular function $f : [0,1]^n \to \mathbb{R}_+$ subject to multiple packing constraints $\mathbb{A}\vec{x} \leq \vec{1}$, where $\mathbb{A} \in \mathbb{R}_+^{m\times n}$ with the following guarantees:*

- *The algorithm is deterministic if provided oracle access for evaluating f and its gradient ∇f;*
- *The algorithm achieves an approximation guarantee of $1 - 1/e - \epsilon$;*
- *The number of rounds of adaptivity and evaluations of f and ∇f are $O\left(\frac{\log(n/\epsilon)\log(m)}{\epsilon^2}\right)$.*

Our packing algorithms are **NC** algorithms and the number of parallel rounds matches the currently best parallel algorithm for linear packing [23]. Along with [1], the result of [23] was the first improvement in decades for solving packing LPs in parallel since the original work of Luby and Nisan [22]. As shown in Figure 1, our algorithm is nearly identical to the linear packing algorithm of [23], thus suggesting that our algorithm's performance improves if the objective has additional structure.

Theorem 3. *For every $\epsilon > 0$, there is an algorithm for maximizing a general (non-monotone) DR-submodular function $f : [0,1]^n \to \mathbb{R}_+$ subject to multiple packing constraints $\mathbb{A}\vec{x} \leq \vec{1}$, where $\mathbb{A} \in \mathbb{R}_+^{m\times n}$ with the following guarantees:*

- *The algorithm is deterministic if provided oracle access for evaluating f and its gradient ∇f;*
- *The algorithm achieves an approximation guarantee of $1/e - \epsilon$;*
- *The number of rounds of adaptivity and evaluations of f and ∇f are $O\left(\frac{\log(n/\epsilon)\log(1/\epsilon)\log(m+n)}{\epsilon^2}\right)$.*

The approximation guarantee of our algorithm nearly matches the best approximation known for general submodular maximization in the sequential setting, which is $0.385 \approx 1/e + 0.0171$ [11]. Prior to our work, the best result for non-monotone submodular maximization is a $1/(2e)$-approximation for a cardinality constraint [3]. No previous result was known even for a single packing constraint.

2 PRELIMINARIES

Let $f : [0,1]^n \to \mathbb{R}_+$ be a non-negative function. The function is *diminishing returns submodular* (DR-submodular) if $\forall \vec{x} \leq \vec{y} \in [0,1]^n$ (where $\leq$ is coordinate-wise), $\forall i \in [n]$, $\forall \delta \in [0,1]$ such that $\vec{x} + \delta\vec{1}_i$ and $\vec{y} + \delta\vec{1}_i$ are still in $[0,1]^n$, it holds

$$f(\vec{x} + \delta\vec{1}_i) - f(\vec{x}) \geq f(\vec{y} + \delta\vec{1}_i) - f(\vec{y}),$$

where $\vec{1}_i$ is the i-th basis vector, i.e., the vector whose i-th entry is 1 and all other entries are 0.

If f is differentiable, f is DR-submodular if and only if $\nabla f(\vec{x}) \geq \nabla f(\vec{y})$ for all $\vec{x} \leq \vec{y} \in [0,1]^n$. If f is twice-differentiable, f is

procedure LinearPacking
  $\eta \leftarrow \frac{\epsilon}{2 \ln m}$
  $\vec{x}_i \leftarrow \frac{\epsilon}{n \|\mathbb{A}_{:i}\|_\infty} \forall i \in [n]$
  while $f(\vec{x}) \leq (1 - O(\epsilon))M$ **do**
    $\vec{c}_i \leftarrow \nabla_i f(\vec{x})$
    $\vec{m}_i \leftarrow \max\left\{\left(1 - M \cdot \frac{(\mathbb{A}^\top \nabla \text{smax}_\eta(\mathbb{A}\vec{x}))_i}{\vec{c}_i}\right), 0\right\}$
    $\vec{d} \leftarrow \eta \vec{x} \circ \vec{m}$
    $\vec{x} \leftarrow \vec{x} + \vec{d}$
  end while
end procedure

procedure SubmodPacking
  $\eta \leftarrow \frac{\epsilon}{2(2+\ln m)}$
  $\vec{x}_i \leftarrow \frac{\epsilon}{n \|\mathbb{A}_{:i}\|_\infty} \quad \forall i \in [n]$
  while $f(\vec{x}) \leq (1 - \exp(-1 + O(\epsilon)))M$ **do**
    $\lambda \leftarrow M - (1 + \eta) f(\vec{x})$
    $\vec{c}_i \leftarrow \nabla_i f((1 + \eta)\vec{x})$
    $\vec{m}_i \leftarrow \max\left\{\left(1 - \lambda \cdot \frac{(\mathbb{A}^\top \nabla \text{smax}_\eta(\mathbb{A}\vec{x}))_i}{\vec{c}_i}\right), 0\right\}$
    $\vec{d} \leftarrow \eta \vec{x} \circ \vec{m}$
    $\vec{x} \leftarrow \vec{x} + \vec{d}$
  end while
end procedure

Figure 1: The algorithm on the left is the algorithm of Mahoney *et al.* [23] for maximizing a linear function $f(\vec{x}) = \langle \vec{c}, \vec{x} \rangle$. The algorithm on the right is our algorithm for monotone DR-submodular maximization. In both algorithms, M is an approximate optimal solution value: $M \leq f(\vec{x}^*) \leq (1 + \epsilon)M$.

DR-submodular if and only if all the entries of the Hessian are *non-positive*, i.e., $\frac{\partial^2 f}{\partial x_i \partial x_j}(\vec{x}) \leq 0$ for all $i, j \in [n]$.

For simplicity, throughout the paper, we assume that f is differentiable. We assume that we are given black-box access to an oracle for evaluating f and its gradient ∇f. We extend the function f to $\mathbb{R}_+^n$ as follows: $f(\vec{x}) = f(\vec{x} \wedge \vec{1})$, where $(\vec{x} \wedge \vec{1})_i = \min\{x_i, 1\}$.

An example of a DR-submodular function is the multilinear extension of a submodular function g. The multilinear extension is defined as

$$G(\vec{x}) = \mathbb{E}[g(R(\vec{x}))] = \sum_{S \subseteq V} g(S) \prod_{i \in S} \vec{x}_i \prod_{i \in V \setminus S} (1 - \vec{x}_i),$$

where $R(\vec{x})$ is a random subset of V where each $i \in V$ is included independently at random with probability $\vec{x}_i$.

We now define the two problems that we consider.

DR-submodular maximization with a polymatroid constraint.

We consider the problem of maximizing a DR-submodular function subject to a polymatroid constraint: max $f(\vec{x})$ subject to $\vec{x} \in \mathbf{P}$, where $\mathbf{P} = \{\vec{x} \colon \vec{x}(S) \leq r(S) \quad \forall S \subseteq V, \vec{x} \geq 0\}$ and $r : 2^V \to \mathbb{R}_+$ is monotone, submodular, and normalized ($r(\emptyset) = 0$). We use the notation $\vec{x}(S)$ as shorthand for $\sum_{i \in S} \vec{x}_i$, i.e., we interpret $\vec{x}$ as a modular function. When r is the rank function of a matroid, $\mathbf{P}$ is the matroid polytope. We refer the reader to Chapter 5 in [20] for more background on matroids and polymatroids.

This problem generalizes the problem of maximizing the multilinear extension of a submodular function subject to a matroid constraint. Rounding algorithms such as pipage rounding and swap rounding allow us to round without any loss a fractional solution in the matroid polytope, and thus our results imply low-adaptive algorithms for submodular maximization with a matroid constraint.

DR-submodular maximization with packing constraints. We consider the problem of maximizing a DR-submodular function subject to packing constraints $\mathbb{A}\vec{x} \leq \vec{1}$, where $\mathbb{A} \in \mathbb{R}_+^{m \times n}$. The problem generalizes the packing problem with a linear objective and the problem of maximizing the multilinear extension of a submodular set function subject to packing constraints.

Since we can afford an ϵ additive loss in the approximation, we may assume that every non-zero entry $A_{i,j}$ satisfies $\frac{\epsilon}{n} \leq A_{i,j} \leq \frac{n}{\epsilon}$. Moreover, we may assume that the optimal solution $\vec{x}^*$ satisfies $\mathbb{A}\vec{x}^* \leq (1 - \epsilon)\vec{1}$.

Basic notation. We use e.g. $\vec{x} = (\vec{x}_1, \ldots, \vec{x}_n)$ to denote a vector in $\mathbb{R}^n$. We use e.g. $\mathbb{A}$ to denote a matrix in $\mathbb{R}^{m \times n}$. For a vector $\vec{a} \in \mathbb{R}^n$, we let $\mathbb{D}(\vec{a})$ be the $n \times n$ diagonal matrix with diagonal entries a_i, and $\mathbb{D}(\vec{a})^+$ be the pseudoinverse of $\mathbb{D}(\vec{a})$, i.e., the diagonal matrix with entries $1/a_i$ if $a_i \neq 0$ and 0 otherwise.

We use the following vector operations: $\vec{x} \vee \vec{y}$ is the vector whose i-th coordinate is $\max\{x_i, y_i\}$; $\vec{x} \wedge \vec{y}$ is the vector whose i-th coordinate is $\min\{x_i, y_i\}$; $\vec{x} \circ \vec{y}$ is the vector whose i-th coordinate is $x_i \cdot y_i$. We write $\vec{x} \leq \vec{y}$ to denote that $\vec{x}_i \leq \vec{y}_i$ for all $i \in [n]$. Let $\vec{0}$ (resp. $\vec{1}$) be the n-dimensional all-zeros (resp. all-ones) vector. Let $\vec{1}_S \in \{0, 1\}^V$ denote the indicator vector of $S \subseteq V$, i.e., the vector that has a 1 in entry i if and only if $i \in S$.

We let $\mathbb{I}$ be the identity matrix. For two matrices $\mathbb{A}$ and $\mathbb{B}$, we write $\mathbb{A} \preceq \mathbb{B}$ to denote that $\mathbb{B} - \mathbb{A} \succeq 0$, i.e., $\mathbb{B} - \mathbb{A}$ is positive semidefinite.

We will use the following result that was shown in previous work [13].

Lemma 4 ([13], Lemma 7). *Let $f : [0, 1]^n \to \mathbb{R}_+$ be a DR-submodular function. For all $\vec{x}^* \in [0, 1]^n$ and $\vec{x} \in [0, 1]^n$, $f(\vec{x}^* \vee \vec{x}) \geq (1 - \|\vec{x}\|_\infty) f(\vec{x}^*)$.*

The softmax function. Let $\eta \in \mathbb{R}_+$ and $\text{smax}_\eta : \mathbb{R}_+^m \to \mathbb{R}_+$ be the function $\text{smax}_\eta(\vec{z}) = \eta \ln\left(\sum_{j=1}^m e^{\frac{1}{\eta} z_j}\right)$. Note that $\|\vec{z}\|_\infty \leq \text{smax}_\eta(\vec{z}) \leq \eta \ln m + \|\vec{z}\|_\infty$. We use $\nabla \text{smax}_\eta(\vec{z})$ to denote the gradient of smax_η, i.e., $\nabla_j \text{smax}_\eta(\vec{z}) = \frac{\partial \text{smax}_\eta(\vec{z})}{\partial z_j} = \frac{e^{\frac{1}{\eta} z_j}}{\sum_{\ell=1}^m e^{\frac{1}{\eta} z_\ell}}$.

We will use the following results that quantify the change in softmax due to an update. Similar results have been proved in the previous work of Mahoney *et al.* [23]. The proof can be found in the full version of this paper.

Lemma 5. *Let $\vec{x}, \vec{d} \in \mathbb{R}_+^m$, and $\mathbb{A} \in \mathbb{R}_+^{m \times n}$. If $\frac{1}{\eta}\|\mathbb{A}\vec{d}\|_\infty \le 1/2$, then*

$$\operatorname{smax}_\eta\left(\mathbb{A}(\vec{x}+\vec{d})\right) \le \operatorname{smax}_\eta\left(\mathbb{A}\vec{x}\right) + \left\langle \mathbb{A}^\top \nabla \operatorname{smax}_\eta(\mathbb{A}\vec{x}), \vec{d} + \|\mathbb{A}\vec{x}\|_\infty \cdot 1/\eta \cdot \mathbb{D}(\vec{x})^+ \cdot (\vec{d} \circ \vec{d})\right\rangle .$$

This immediately yields a very useful corollary.

Corollary 6. *Suppose that $\|A\vec{x}\|_\infty \le 1$, under the same conditions from Lemma 5. Letting $\vec{d} = \eta\mathbb{M}\vec{x}$, where $\mathbb{M}$ is a diagonal matrix such that $\mathbb{M} \preceq \mathbb{I}$, one has that*

$$\operatorname{smax}_\eta\left(\mathbb{A}(\vec{x}+\vec{d})\right) \le \operatorname{smax}_\eta\left(\mathbb{A}\vec{x}\right) + \eta\left\langle \mathbb{A}^\top \nabla \operatorname{smax}_\eta(\mathbb{A}\vec{x}), \mathbb{M}\vec{x} + \mathbb{M}^2\vec{x}\right\rangle$$

Furthermore, given $\lambda \in \mathbb{R}_+, \vec{c} \in \mathbb{R}_+^n$, and letting

$$\mathbb{M}_{ii} = \left(1 - \lambda \cdot \frac{\left(\mathbb{A}^\top \nabla \operatorname{smax}_\eta(\mathbb{A}\vec{x})\right)_i}{\vec{c}_i}\right) \vee 0 ,$$

one has that

$$\frac{\operatorname{smax}_\eta\left(\mathbb{A}(\vec{x}+\vec{d})\right) - \operatorname{smax}_\eta\left(\mathbb{A}\vec{x}\right)}{\langle \vec{c}, \vec{d}\rangle} \le \frac{1}{\lambda} .$$

3 MONOTONE MAXIMIZATION WITH A MATROID CONSTRAINT

In this section, we consider the problem of maximizing a monotone DR-submodular function subject to a polymatroid constraint: $\max f(\vec{x})$ subject to $\vec{x} \in \mathbf{P}$, where $\mathbf{P} = \{\vec{x} : \vec{x}(S) \le r(S) \quad \forall S \subseteq V, \vec{x} \ge 0\}$ and $r : 2^V \to \mathbb{R}_+$ is monotone, submodular, and normalized ($r(\emptyset) = 0$). For $\alpha \in [0,1]$, we use $\alpha\mathbf{P}$ to denote the set $\alpha\mathbf{P} = \{\alpha\vec{x} : \vec{x} \in \mathbf{P}\}$. We use $\vec{x}^*$ to denote an optimal solution to $\max_{\vec{x}\in\mathbf{P}} f(\vec{x})$.

Our algorithm is shown in Algorithm 1. The algorithm requires an $(1+\epsilon)$ approximation to the optimum value, more precisely, a value M such that $M \le f(\vec{x}^*) \le (1+\epsilon)M$. An n-approximation to $f(\vec{x}^*)$ is $M_0 = \max_{i\in[n]} f(\vec{1}_i)$. Given this value, we can try $2\ln n/\epsilon$ guesses for M: $M_0, (1+\epsilon)M_0, (1+\epsilon)^2 M_0, \ldots$ in parallel and return the best solution from all the guesses.

In this section, we assume that, for every vector $\vec{x}$, the entries of the gradient $\nabla f(\vec{x})$ are at most DM, where $D = \text{poly}(n/\epsilon)$. This assumption is satisfied when f is the multilinear extension of a submodular function, since $\nabla_i f(\vec{x}) \le f(\vec{1}_i) \le f(\vec{x}^*) \le (1+\epsilon)M$.

Our algorithm and analysis for *non-monotone* maximization subject to a polymatroid constraint is an extension of the monotone case, and it is given in Appendix 6.

High level overview of the approach. The starting point of our algorithm is the continuous Greedy approach for submodular maximization. Algorithms based on continuous Greedy are iterative algorithms that increase the solution over time. In each iteration, the algorithms take the linear approximation given by the gradient at the current solution, and find a base of the matroid that maximizes this linear approximation. The optimum linear-weight base is given by the Greedy algorithm that considers the elements in decreasing order according to the weights and adds the current element if it is feasible to do so. Given this base $\vec{b}$, the algorithms perform the update $\vec{x} \leftarrow \vec{x} + \eta\vec{b}$, where η is an appropriately chosen step size.

As mentioned in the introduction, there are two important points to note about the above iterative schemes: (1) since the gradient changes very quickly, the step size η needs to be very small to ensure a good approximation guarantee, and (2) the updates increase the coordinates of the solution by small amounts and we need polynomially many iterations to converge.

We overcome the above difficulties as follows. By choosing our update vector very carefully, we ensure that the coordinates of the solution are increasing *multiplicatively*, and the algorithm converges in only a *poly-logarithmic* number of iterations. The idea of using multiplicative updates is reminiscent of the work of Luby and Nisan for solving LPs in parallel, but it cannot be implemented directly in the submodular setting, since the linear approximation given by the gradient is changing too quickly. Our key idea here is that, instead of using the gradient at the current solution, we use the gradient at $(1+\epsilon)\vec{x}$, which is an upper bound on the solution *after* the multiplicative update. This strikes the right balance between how large the step size is and how much we are underestimating the gain.

We now briefly discuss the algorithm and in particular how to construct the update vectors. In order to obtain the nearly-optimal $1-1/e-\epsilon$ approximation, the algorithm builds the solution over $1/\epsilon$ epochs (iterations of the outer for loop), and each epoch decreases by an ϵ factor the distance between the optimal value and the current solution value. (The reader may find it helpful to first consider the variant of our algorithm with a single epoch, which leads to a $1/2-\epsilon$ approximation.) In a given epoch, the algorithm iteratively updates the solution as follows. We first compute the gradient at the future point (line 8). To ensure that we are updating the most valuable coordinates, we bucket the gradient values of the coordinates that can be increased into logarithmically many buckets, and we update each bucket in turn as shown on lines 12–16.

A key difficulty in the analysis is to show that the above updates increase the solution very fast while at the same time the function value gain is proportional to the optimal solution. To this end, we use the structure of the polymatroid constraint to construct an evolving solution based on $\vec{x}^*$ and our current solution (see Lemma 7 and the solutions $\vec{o}^{(t)}$ defined below). We use a subtle charging argument to relate the solution gain after each update to this evolving solution and to relate it to the optimum solution. We also use the structure of the tight sets of the polymatroid to show that the solution is increasing very fast and the algorithm terminates in a poly-logarithmic number of iterations (see Lemmas 8 and 13).

Analysis of the approximation guarantee. We will use the following lemma in the analysis of the approximation guarantee of Algorithm 1. We drop the vector notation for notational simplicity.

Lemma 7. *Consider three vectors a, b, c such that $a + c \in \mathbf{P}$, $b \in \mathbf{P}$, and $a \le b$. There exists a vector d such that $0 \le d \le c$, $b + d \in \mathbf{P}$, and $\|c - d\|_1 \le \|b - a\|_1$.*

Proof. We let e_i denote the i-th basis vector, i.e., the n-dimensional vector whose i-th entry is 1 and all other entries are 0. For a vector $x \in \mathbf{P}$, we say that a set $S \subseteq V$ is *x-tight* if $x(S) = r(S)$. The submodularity of r implies that, if S and T are x-tight then $S \cup T$ and $S \cap T$ are also x-tight. Thus, for every element $u \in V$, there is a unique minimal x-tight set that contains u.

Let $\hat{b} = a$ and $\hat{d} = c$. We will iteratively increase $\hat{b}$ and decrease $\hat{d}$ until $\hat{b}$ becomes equal to b; at that point, the vector $\hat{d}$ will be

Algorithm 1 Algorithm for monotone maximization with a polymatroid constraint.

M is an approximate optimal solution value: $M \leq f(\vec{x}^*) \leq (1+\epsilon)M$, where $\vec{x}^* \in \operatorname{argmax}_{\vec{x}\in\mathbf{P}} f(\vec{x})$, D is chosen such that $\|\nabla f(\vec{x})\|_\infty \leq MD$.
$\vec{z} \leftarrow 0$
for $j \leftarrow 1$ to $1/\epsilon$ **do**
  $\vec{x}^{(0)} \leftarrow \frac{\epsilon^2}{nD}\vec{1}$
  $t \leftarrow 0$
  Let $g(\vec{x}) = f(\vec{x}+\vec{z})$
  while $g(\vec{x}^{(t)}) - g(\vec{x}^{(0)}) \leq \epsilon((1-10\epsilon)M - g(\vec{x}^{(0)}))$ **do**
    $\vec{c}_i \leftarrow \nabla_i g((1+\epsilon)\vec{x}^{(t)})$
    Let $T(\vec{x})$ for $\vec{x} \in \frac{\epsilon}{1+\epsilon}\mathbf{P}$ be the *maximal* set S such that $\vec{x}(S) = \frac{\epsilon}{1+\epsilon}r(S)$.
    Let $v_1 = \max_{i\notin T(\vec{x}^{(t)})} \vec{c}_i$ and v_2 be the maximum power of $1+\epsilon$ such that $v_2 \leq v_1$.
    $\vec{y} \leftarrow 0$
    for i from 1 to n **do**
      if $\vec{c}_i \geq v_2$ **then**
        Let $\vec{y}_i$ be the maximum value such that $\vec{y}_i \leq \epsilon\vec{x}_i^{(t)}$ and $(1+\epsilon)(\vec{x}^{(t)}+\vec{y}) \in \epsilon\mathbf{P}$.
      end if
    end for
    $\vec{x}^{(t+1)} \leftarrow \vec{x}^{(t)} + \vec{y}$
    $t \leftarrow t+1$
  end while
  $\vec{z} \leftarrow \vec{z} + \vec{x}^{(t)}$
end for
return $\vec{z}$

the desired vector d. We will maintain the following invariants: $\hat{b} + \hat{d} \in \mathbf{P}$, $\hat{d} \geq 0$, $\hat{b}$ can only increase, $\hat{d}$ can only decrease, and the total amount by which the coordinates of $\hat{b}$ increase is at least the total amount by which the coordinates of $\hat{d}$ decrease.

The update procedure is as follows. Let i be a coordinate such that $\hat{b}_i < b_i$. Let $\delta \geq 0$ be the maximum amount such that $\hat{b} + \delta e_i + \hat{d} \in \mathbf{P}$. We increase $\hat{b}_i$ by $\min\{\delta, b_i - \hat{b}_i\}$. If $\hat{b}_i$ reaches b_i, we are done with this coordinate and we can move on to the next coordinate that needs to be increased. Otherwise, there is a $(\hat{b}+\hat{d})$-tight set that contains i. Let T be the minimal $(\hat{b}+\hat{d})$-tight set that contains i. Since $b \in \mathbf{P}$, T contains a coordinate j for which $\hat{d}_j > 0$: since $b \geq \hat{b}$ and $b_i > \hat{b}_i$, we have $b(T) > \hat{b}(T)$; since b is feasible and T is $(\hat{b}+\hat{d})$-tight, we have $b(T) \leq r(T) = \hat{b}(T) + \hat{d}(T)$. Let $j \in T$ be such that $\hat{d}_j > 0$. Let $\gamma > 0$ be the maximum amount such that $\hat{b} + \hat{d} + \gamma e_i - \gamma e_j \in \mathbf{P}$. Let $\delta = \min\{b_i - \hat{b}_i, \gamma, \hat{d}_j\}$. We update $\hat{b}$ and $\hat{d}$ as follows: we increase coordinate i in $\hat{b}$ by δ, and we decrease coordinate j in $\hat{d}$ by δ. Note that this update maintains the desired invariants. We repeat this procedure until $\hat{b}_i$ becomes equal to b_i. Note that after each step where we increase $\hat{b}$ and decrease $\hat{d}$, either 1) $\delta = b_i - \hat{b}_i$, or 2) the minimal tight set of $\hat{b}+\hat{d}$ containing i shrinks (the case $\delta = \gamma$) or 3) one coordinate of $\hat{d}$ becomes 0 (the case $\delta = \hat{d}_j$) so the procedure finishes in a finite number of steps.

When the update procedure terminates, we have $\hat{b} = b$ and we let $d = \hat{d}$. It follows from the invariants above that d has the desired properties. □

We now show that the algorithm achieves a $1 - 1/e - \epsilon$ approximation guarantee. We consider each iteration of the algorithm and we analyze the increase in value when updating $\vec{x}^{(t)}$ to $\vec{x}^{(t+1)}$. Using Lemma 7, we show that we can define a sequence of vectors $\vec{o}^{(t)}$ based on $\vec{x}^{(t)}$ and the optimal solution that allows us to relate the gain of the algorithm to the optimum value. To this end, consider iteration j of the outer for loop. We define a vector $\vec{o}^{(t)}$ for each iteration t of the while loop as follows. Let $\vec{x}^{(-1)} = 0$ and $\vec{o}^{(-1)} = \vec{z} \vee \vec{x}^* - \vec{z}$; note that $\vec{o}^{(-1)} \in \mathbf{P}$. Suppose we have already defined a vector $\vec{o}^{(t)}$ such that $\vec{x}^{(t)} + \frac{\epsilon}{1+\epsilon}\vec{o}^{(t)} \in \frac{\epsilon}{1+\epsilon}\mathbf{P}$. We define $\vec{o}^{(t+1)}$ to be the vector d guaranteed by Lemma 7 for $a = \frac{1+\epsilon}{\epsilon}\vec{x}^{(t)}$, $b = \frac{1+\epsilon}{\epsilon}\vec{x}^{(t+1)}$, $c = \vec{o}^{(t)}$. By Lemma 7, the vector $\vec{o}^{(t+1)}$ has the following properties:

- (P_1) $\vec{x}^{(t+1)} + \frac{\epsilon}{1+\epsilon}\vec{o}^{(t+1)} \in \frac{\epsilon}{1+\epsilon}\mathbf{P}$,
- (P_2) $0 \leq \vec{o}^{(t+1)} \leq \vec{o}^{(t)}$,
- (P_3) $\frac{\epsilon}{1+\epsilon}\|\vec{o}^{(t)} - \vec{o}^{(t+1)}\|_1 \leq \|\vec{x}^{(t+1)} - \vec{x}^{(t)}\|_1$,
- (P_4) $\operatorname{support}(o^{(t+1)}) \subseteq V \setminus T(\vec{x}^{(t+1)})$ by (P_1), where the support is the set of non-zero coordinates.

We now use these properties to relate the algorithm's gain to that of $\vec{z} \vee \vec{x}^* - \vec{z}$. We start with the following observations. Recall that we are considering a fixed iteration j of the outer for loop, and t indexes the iterations of the while loop in the current iteration j.

Lemma 8. *We have*

(a) *For every $\vec{x} \in \frac{\epsilon}{1+\epsilon}\mathbf{P}$, there is a unique maximal set S satisfying $\vec{x}(S) = \frac{\epsilon}{1+\epsilon}r(S)$.*
(b) *For every t, we have $T(\vec{x}^{(t)}) \subseteq T(\vec{x}^{(t+1)})$.*
(c) *The values v_1 and v_2 are non-increasing over time.*

Proof. (a) Since r is submodular and $\vec{x}$ is modular, the set $\{S \subseteq V : \vec{x}(S) = \frac{\epsilon}{1+\epsilon}r(S)\}$ is closed under intersection and union, and thus it has a unique maximal set.
(b) Since $x^{(t+1)} \geq x^{(t)}$, $T(\vec{x}^{(t)})$ remains tight with respect to $\vec{x}^{(t+1)}$, i.e., $\vec{x}^{(t+1)}(T(\vec{x}^{(t)})) = \frac{\epsilon}{1+\epsilon}r(T(\vec{x}^{(t)}))$. Thus the maximality and uniqueness of $T(\vec{x}^{(t+1)})$ imply that $T(\vec{x}^{(t)}) \subseteq T(\vec{x}^{(t+1)})$.
(c) Since g is DR-submodular and $\vec{x}^{(t)} \leq \vec{x}^{(t+1)}$, we have $\nabla g((1+\epsilon)\vec{x}^{(t)}) \geq \nabla g((1+\epsilon)\vec{x}^{(t+1)})$. Additionally, $T(\vec{x}^{(t)}) \subseteq T(\vec{x}^{(t+1)})$. Thus v_1 is non-increasing and therefore v_2 is non-increasing. □

We note that the gradient is non-negative when the function is monotone. We state the lemmas with $\nabla g(\cdot) \vee \vec{0}$ so that they apply to both monotone and non-monotone functions, as we will reuse them for non-monotone maximization.

Lemma 9. *We have*

$$g(\vec{x}^{(t+1)}) - g(\vec{x}^{(t)}) \geq \frac{\epsilon(1-\epsilon)}{1+\epsilon}\left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}) \vee \vec{0}, \vec{o}^{(t)} - \vec{o}^{(t+1)} \right\rangle.$$

Proof. We have

$$\begin{aligned}
&g(\vec{x}^{(t+1)}) - g(\vec{x}^{(t)}) \overset{(1)}{\geq} \langle \vec{y}, \nabla g(\vec{x}^{(t+1)}) \rangle \\
&\overset{(2)}{\geq} \langle \vec{y}, \nabla g((1+\epsilon)\vec{x}^{(t)}) \rangle \\
&\overset{(3)}{\geq} \|\vec{y}\|_1 v_2 \\
&\overset{(4)}{\geq} \|\vec{y}\|_1 v_1 (1-\epsilon) \\
&\overset{(5)}{\geq} (1-\epsilon) \frac{\|\vec{y}\|_1}{\|\vec{o}^{(t)} - \vec{o}^{(t+1)}\|_1} \left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}) \vee \vec{0}, \vec{o}^{(t)} - \vec{o}^{(t+1)} \right\rangle \\
&\overset{(6)}{\geq} (1-\epsilon) \frac{\epsilon}{1+\epsilon} \left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}) \vee \vec{0}, \vec{o}^{(t)} - \vec{o}^{(t+1)} \right\rangle ,
\end{aligned}$$

where (1) holds by concavity along non-negative directions, (2) is due to $\vec{x}^{(t+1)} \leq (1+\epsilon)\vec{x}^{(t)}$ and gradient monotonicity, (3) and (4) are due to the choice of $\vec{y}$, v_2, and v_1, (6) is due to property (P_3).

We can show (5) as follows. By property (P_4), we have $\text{support}(\vec{o}^{(t)}) \subseteq V \setminus T(\vec{x}^{(t)})$. Thus we have $\nabla_i g((1+\epsilon)\vec{x}^{(t)}) \leq v_1$ for all $i \in \text{support}(\vec{o}^{(t)})$. By property (P_2), we have $\vec{o}^{(t+1)} \leq \vec{o}^{(t)}$.

Thus

$$\left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}), \vec{o}^{(t)} - \vec{o}^{(t+1)} \right\rangle \leq v_1 \|\vec{o}^{(t)} - \vec{o}^{(t+1)}\|_1 .$$

□

By repeatedly applying Lemma 9, we obtain the following lemma.

Lemma 10. *We have*

$$\begin{aligned}
&g(\vec{x}^{(t+1)}) - g(\vec{x}^{(0)}) \\
&\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} \left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}) \vee \vec{0}, \vec{o}^{(0)} - \vec{o}^{(t+1)} \right\rangle \\
&\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} \left(g(\vec{o}^{(0)} - \vec{o}^{(t+1)} + (1+\epsilon)\vec{x}^{(t)}) - g((1+\epsilon)\vec{x}^{(t)}) \right) .
\end{aligned}$$

Proof. By Lemma 9 and DR-submodularity, we have

$$\begin{aligned}
&g(\vec{x}^{(t+1)}) - g(\vec{x}^{(0)}) \\
&\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} \sum_{j=0}^{t} \left\langle \nabla g((1+\epsilon)\vec{x}^{(j)}) \vee \vec{0}, \vec{o}^{(j)} - \vec{o}^{(j+1)} \right\rangle \\
&\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} \sum_{j=0}^{t} \left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}) \vee \vec{0}, \vec{o}^{(j)} - \vec{o}^{(j+1)} \right\rangle \\
&= \frac{\epsilon(1-\epsilon)}{1+\epsilon} \left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}) \vee \vec{0}, \vec{o}^{(0)} - \vec{o}^{(t+1)} \right\rangle \\
&\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} \left(g(\vec{o}^{(0)} - \vec{o}^{(t+1)} + (1+\epsilon)\vec{x}^{(t)}) - g((1+\epsilon)\vec{x}^{(t)}) \right) .
\end{aligned}$$

□

Lemma 10 implies that every iteration of the while loop increases at least one coordinate, and thus the while loop eventually terminates.

Lemma 11. *In every iteration t, we have $T(\vec{x}^{(t)}) \neq V$, i.e., some coordinate increases in each iteration.*

Proof. Suppose that $\vec{x}^{(t)}(V) = \frac{\epsilon}{1+\epsilon} r(V)$. By properties (P_1) and (P_2), we have $\vec{o}^{(t+1)} = 0$. By Lemma 10 and monotonicity, we have

$$\begin{aligned}
&g(\vec{x}^{(t+1)}) - g(\vec{x}^{(0)}) \\
&\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} \left(g(\vec{o}^{(0)} + (1+\epsilon)\vec{x}^{(t)}) - g((1+\epsilon)\vec{x}^{(t)}) \right) \\
&\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} (g(\vec{x}^{(0)} + \vec{o}^{(0)}) - g((1+\epsilon)\vec{x}^{(t)})) \\
&\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} \left(f(\vec{z} \vee \vec{x}^*) - 2D\|\vec{x}^{(0)}\|_1 - g((1+\epsilon)\vec{x}^{(t)}) \right) .
\end{aligned}$$

In the last inequality, we used the fact that $\|\vec{x}^{(0)} + \vec{o}^{(0)} - \vec{o}^{(-1)}\|_1 \leq 2\|\vec{x}^{(0)}\|_1$ (by Lemma 7). By observing $g((1+\epsilon)\vec{x}^{(t)}) \leq (1+\epsilon)g(\vec{x}^{(t)})$ and adding $\epsilon(1-\epsilon)(g(\vec{x}^{(t)}) - g(\vec{x}^{(0)}))$ to both sides, we obtain:

$$\begin{aligned}
&(1+\epsilon(1-\epsilon))(g(\vec{x}^{(t+1)}) - g(\vec{x}^{(0)})) \\
&\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} \left(f(\vec{z} \vee \vec{x}^*) - 2D\|\vec{x}^{(0)}\|_1 - (1+\epsilon)g(\vec{x}^{(0)}) \right) .
\end{aligned}$$

By monotonicity, $f(\vec{z} \vee \vec{x}^*) \geq f(\vec{x}^*)$, and we also have $2D\|\vec{x}^{(0)}\|_1 \leq 2\epsilon^2 M$; $\epsilon g(\vec{x}^{(0)}) \leq 2\epsilon M$. Thus the gain is large enough for the while loop to terminate. □

Thus the algorithm terminates. Finally, we show that the solution returned is a $1 - 1/e - O(\epsilon)$ approximation.

Lemma 12. *The solution $\vec{z}$ returned by Algorithm 1 is feasible and it satisfies $f(\vec{z}) \geq (1 - 1/e - O(\epsilon))M \geq (1 - 1/e - O(\epsilon))f(\vec{x}^*)$.*

Proof. For each iteration j of the outer for loop, let $\vec{z}^{(j)}$ be the solution $\vec{z}$ at the beginning of the iteration. Consider an iteration j. In each iteration t of the while loop, we have $\vec{x}^{(t)} \in \epsilon\mathbf{P}$, and thus $\vec{z}^{(j+1)} - \vec{z}^{(j)} \in \epsilon\mathbf{P}$. Since there are $1/\epsilon$ iterations, the final solution $\vec{z}$ is in $\mathbf{P}$.

We now analyze the approximation guarantee. For each iteration j, the terminating condition of the while loop guarantees that

$$f(\vec{z}^{(j+1)}) - f(\vec{z}^{(j)}) \geq \epsilon((1-10\epsilon)M - f(\vec{z}^{(j)})) .$$

By rearranging, we obtain

$$(1-10\epsilon)M - f(\vec{z}^{(j+1)}) \leq (1-\epsilon)((1-10\epsilon)M - f(\vec{z}^{(j)})) .$$

Thus, by induction,

$$(1-10\epsilon)M - f(\vec{z}^{(1/\epsilon)}) \leq (1-\epsilon)^{1/\epsilon}(1-10\epsilon)M,$$

and thus we obtain a $1 - 1/e - O(\epsilon)$ approximation. □

Analysis of the number of iterations. We now upper bound the total number of iterations of Algorithm 1, and thus the number of rounds of adaptivity.

Lemma 13. *The total number of iterations and rounds of adaptivity is $O(\log^2 n/\epsilon^3)$.*

Proof. Consider an iteration j of the outer for loop. Recall that the values v_1 and v_2 are non-increasing over time, the solutions $\vec{x}^{(t)}$ are non-decreasing, the gradient values $\vec{c}$ are non-increasing (by DR-submodularity), and the sets $T(\vec{x}^{(t)})$ can only gain coordinates (by Lemma 8).

Let us now divide the iterations of the while loop into phases, where a phase is comprised of the iterations with the same value v_2.

Claim 14. *There are $O(\log n/\epsilon)$ iterations in a phase.*

Proof. Over the iterations of a phase, the set

$$\{i : i \notin T(\vec{x}^{(t)}) \text{ and } \vec{c}_i \geq v_2\}$$

cannot gain new coordinates. Additionally, each iteration of a phase increases at least one coordinate. Thus the coordinate i that is increased in the last iteration of the phase is increased in all of the iterations of the phase. Each iteration of the phase, except possibly the last iteration, increases coordinate i by a multiplicative $(1+\epsilon)$ factor (if we have $\vec{y}_i < \epsilon \vec{x}_i^{(t)}$ in some iteration t, after the update we cannot increase coordinate i anymore and $i \in T(\vec{x}^{(t+1)})$). We can only increase a coordinate $O(\log n/\epsilon)$ times before the solution goes out of $\mathbf{P}$. Thus the phase has $O(\log n/\epsilon)$ iterations. □

Claim 15. *The number of phases is $O(\log(n/\epsilon)/\epsilon)$.*

Proof. As noted earlier, the value v_2 is non-increasing over time. Our assumption on the gradient entries guarantees that $v_2 \leq \text{poly}(n/\epsilon)M$. We now show that $v_2 \geq \text{poly}(\epsilon/n)M$, since otherwise the terminating condition of the while loop is satisfied. Suppose that $v_2 \leq \frac{\epsilon^2}{n}M$. Since the support of $\vec{o}^{(t+1)}$ is contained in $V \setminus T(\vec{x}^{(t)})$ (by properties P_2 and P_4), we have

$$\left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}), \vec{o}^{(t+1)} \right\rangle \leq (1+\epsilon)v_2 n \leq (1+\epsilon)\epsilon^2 M\,.$$

By DR-submodularity and monotonicity, we have

$$\begin{aligned}\left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}), \vec{o}^{(0)} \right\rangle &\geq g((1+\epsilon)\vec{x}^{(t)} + \vec{o}^{(0)}) - g((1+\epsilon)\vec{x}^{(t)}) \\ &\geq g(\vec{x}^{(0)} + \vec{o}^{(0)}) - g((1+\epsilon)\vec{x}^{(t)})\,.\end{aligned}$$

By Lemma 10 and the above inequalities,

$$\begin{aligned} g(\vec{x}^{(t+1)}) - g(\vec{x}^{(0)}) &\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon}\left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}) \vee \vec{0}, \vec{o}^{(0)} - \vec{o}^{(t+1)} \right\rangle \\ &\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon}\left(g(\vec{x}^{(0)} + \vec{o}^{(0)}) - g((1+\epsilon)\vec{x}^{(t)}) - (1+\epsilon)\epsilon^2 M\right) \\ &\geq \frac{\epsilon(1-\epsilon)}{1+\epsilon}\Big(f(\vec{z} \vee \vec{x}^*) - 2D\|\vec{x}^{(0)}\|_1 - g((1+\epsilon)\vec{x}^{(t)}) \\ &\qquad - (1+\epsilon)\epsilon^2 M\Big)\,. \end{aligned}$$

In the last inequality, we used the fact that $\|\vec{x}^{(0)} + \vec{o}^{(0)} - \vec{o}^{(-1)}\|_1 \leq 2\|\vec{x}^{(0)}\|_1$ (by Lemma 7). By monotonicity, $f(\vec{z} \vee \vec{x}^*) \geq f(\vec{x}^*)$, and thus the gain is large enough for the while loop to terminate.

To summarize, we have $\text{poly}(\epsilon/n)M \leq v_2 \leq \text{poly}(n/\epsilon)M$, and thus there are $O(\log(n/\epsilon)/\epsilon)$ different values of v_2. □

Therefore the total number of iterations is $O(1/\epsilon) \cdot O(\log n/\epsilon) \cdot O(\log(n/\epsilon)/\epsilon) = O(\log^2 n/\epsilon^3)$. □

4 MONOTONE MAXIMIZATION WITH PACKING CONSTRAINTS

Our algorithm for monotone DR-submodular maximization subject to packing constraints is shown in Algorithm 2. Similarly to the algorithm for a matroid constraint, the algorithm requires an $(1+\epsilon)$ approximation to the optimum value. We obtain the value M by guessing as before. If in some iteration of the algorithm the update vector $\vec{d}$ on line 8 is equal to 0, the guessed value is too high and the algorithm can terminate.

High level overview of the approach. Our algorithm is based on the Lagrangian-relaxation approach developed in the context of solving packing and covering LPs [22, 23, 31], and in particular the algorithm of [23] that achieves the currently best parallel running time. Analogously to [23], our algorithm replaces the hard packing constraints $\mathbb{A}\vec{x} \leq \vec{1}$ (equivalently, $\|\mathbb{A}\vec{x}\|_\infty \leq 1$) with the constraint $\text{smax}(\mathbb{A}\vec{x}) \leq 1$, which is a smooth convex approximation to the original constraint. We can think of the $\text{smax}(\mathbb{A}\vec{x})$ as a potential that measures how much progress the algorithm is making towards satisfying the constraints. The overall approach is to start with a small solution $\vec{x}$ and to iteratively increase it over time, while ensuring the objective value $f(\vec{x})$ increases sufficiently; here time is tracking the softmax potential: $t = \text{smax}(\mathbb{A}\vec{x})$ and t is increasing from 0 to 1. When the objective function is linear, the approach of [23] as well as previous works is the following. In each iteration, the algorithm of [23] picks a subset of the coordinates to update based on the gradient of the softmax function, and it updates the selected coordinates in such a way that the increase in softmax is not too large.

A natural strategy for extending this approach to the submodular setting is to "linearize" the function: compute the gradient at the current solution and use the linear approximation to the function given by the gradient. As before, a key difficulty with this approach is that the gradient is changing very fast and we cannot make large updates. To overcome this difficulty, we use the same strategy as in the matroid case and compute the gradient at a future point. This allows us to make large, *multiplicative* updates to the solution and to converge in a small number of iterations. The resulting algorithm is nearly identical to the linear algorithm (see Figure 1).

While the algorithm is nearly identical to the linear case, our analysis is substantially more involved due to the fact that the linear approximation is changing over time and it is a significant departure from previous works such as [14]. Previous algorithms, such as the linear packing algorithm of [31] and the submodular packing of [14], are divided into phases where, within each phase, the objective value increases by ϵ times the optimal value. This division makes the analysis easier. For instance, the Greedy algorithm would like to pick coordinates whose gain is proportional to the difference between the optimal value and the current solution value. Within each phase, up to an $1+\epsilon$ factor, this threshold is the same. The total saturation of the constraints also behaves similarly. The fact that, up to an $1+\epsilon$ approximation, all relevant quantities are constant is extremely useful in this context because one can adapt the analysis from the case of a linear objective. However, this partition can lead to a suboptimal number of iterations: there are $\Omega(1/\epsilon)$ phases and the number of iterations might be suboptimal by a $\Omega(1/\epsilon)$ factor. Instead, we remove the phases and develop a global argument on the number of iterations. A global argument exists for the linear case [23] but here we need a more general argument with varying selection thresholds over the iterations and varying contributions from different coordinates over the iterations. Intuitively, a coordinate is important if its marginal gain on the current solution is high. We can show that, on aggregate, the coordinates of the optimal solution are important. However, over the iterations, different coordinates might be important at different times. In contrast, in the linear case, the relative importance is exactly the same

over the iterations. Thus, in the linear case, we know that the algorithm keeps increasing, say, the most important coordinate in the optimal solution and that coordinate cannot exceed 1 so the algorithm finishes quickly. In our case, this is not clear because different coordinates are important at different times so the algorithm might increase different coordinates in different iterations and can prolong the process. Nonetheless, because the solution always increases and by the diminishing return property, we know that the importance of all coordinates decreases over time. We use this property to relate the contribution from different iterations and effectively argue that the algorithm cannot keep selecting different coordinates at different times. The precise argument is intricate and requires the division of the iterations into only two parts, instead of $\Omega(1/\epsilon)$ phases (see Lemma 20).

Algorithm 2 Algorithm for $\max_{\vec{x}\in[0,1]^n:\, \mathbb{A}\vec{x}\le(1-\epsilon)\vec{1}} f(\vec{x})$, where f is a non-negative monotone DR-submodular function and $\mathbb{A} \in \mathbb{R}_+^{m\times n}$.

1: $\eta \leftarrow \frac{\epsilon}{2(2+\ln m)}$
2: M is an approximate optimal solution value: $M \le f(\vec{x}^*) \le (1+\epsilon)M$, where $\vec{x}^* \in \operatorname{argmax}_{\vec{x}:\, \mathbb{A}\vec{x}\le(1-\epsilon)\vec{1}} f(\vec{x})$.
3: $\vec{x}_i \leftarrow \frac{\epsilon}{n\|\mathbb{A}_{:i}\|_\infty} \quad \forall i \in [n]$
4: **while** $f(\vec{x}) \le (1-\exp(-1+10\epsilon))M$ **do**
5: $\quad \lambda \leftarrow M - (1+\eta)f(\vec{x})$
6: $\quad \vec{c}_i \leftarrow \nabla_i f((1+\eta)\vec{x}) \quad \forall i \in [n]$
7: $\quad \vec{m}_i \leftarrow \left(1 - \lambda \cdot \frac{(\mathbb{A}^\top \nabla \mathrm{smax}_\eta(\mathbb{A}\vec{x}))_i}{\vec{c}_i}\right) \vee 0$, for all i with $\vec{c}_i \ne 0$, and $\vec{m}_i = 0$ if $\vec{c}_i = 0$
8: $\quad \vec{d} \leftarrow \eta\vec{x} \circ \vec{m}$
9: $\quad \vec{x} \leftarrow \vec{x} + \vec{d}$
10: **end while**

The following lemma shows that every iteration makes progress at the right rate. More precisely, we show that the ratio between the change in the value of f and the change in smax_η is at least equal to the current distance to $\vec{x}^*$ in function value.

Lemma 16. *We have*

$$\frac{f(\vec{x}+\vec{d}) - f(\vec{x})}{\mathrm{smax}_\eta(\mathbb{A}(\vec{x}+\vec{d})) - \mathrm{smax}_\eta(\mathbb{A}\vec{x})} \ge \lambda.$$

Proof. Using Corollary 6 we bound

$$\begin{aligned}\mathrm{smax}_\eta(\mathbb{A}(\vec{x}+\vec{d})) - \mathrm{smax}_\eta(\mathbb{A}\vec{x}) &\le \frac{1}{\lambda}\cdot\langle\nabla f(\vec{x}+\eta\vec{x}), \vec{d}\rangle \\ &\overset{(1)}{\le} \frac{1}{\lambda}(f(\vec{x}+\vec{d}) - f(\vec{x}))\,,\end{aligned}$$

where (1) is due to concavity along the direction of $\vec{d}$ and the fact that $\nabla f(\vec{x}+\eta\vec{x}) \le \nabla f(\vec{x}+\vec{d})$, since $\vec{d} \le \eta\vec{x}$. □

Next we show that every iteration is well defined, in the sense that it performs a nonzero update on the vector $\vec{x}$.

Lemma 17. *In every iteration we have $\vec{d} \ne \vec{0}$.*

Proof. Suppose for contradiction that there is an iteration where $\vec{d} = \vec{0}$. Then for all coordinates $i \in [n]$,

$$\frac{\nabla_i f((1+\eta)\vec{x}))}{\left(\mathbb{A}^\top \nabla \mathrm{smax}_\eta(\mathbb{A}\vec{x}))\right)_i} < \lambda\,.$$

Therefore

$$\begin{aligned}&f(\vec{x}^* \vee (1+\eta)\vec{x}) - f((1+\eta)\vec{x}) \\ &\le \langle\nabla f((1+\eta)\vec{x}), \vec{x}^* \vee (1+\eta)\vec{x} - (1+\eta)\vec{x}\rangle \\ &\overset{(1)}{\le} \langle\nabla f((1+\eta)\vec{x}), \vec{x}^*\rangle \\ &< \lambda\cdot\langle\mathbb{A}^\top\nabla\mathrm{smax}_\eta(\mathbb{A}\vec{x}), \vec{x}^*\rangle \\ &\le \lambda\cdot\|\nabla\mathrm{smax}_\eta(\mathbb{A}\vec{x})\|_1\|\mathbb{A}\vec{x}^*\|_\infty \\ &\overset{(2)}{\le} \lambda(1-\epsilon)\,.\end{aligned}$$

In (1) we used the fact that $(a\vee b) - b \le a$, for $a, b \ge 0$, and in (2) we used $\|\nabla\mathrm{smax}_\eta(\mathbb{A}\vec{x})\|_1 \le 1$, and $\|\mathbb{A}\vec{x}^*\|_\infty \le 1-\epsilon$.

However, from monotonicity we have $f(\vec{x}^*) \le f(\vec{x}^* \vee (1+\eta)\vec{x})$, and from concavity along nonnegative directions, we get that $f((1+\eta)\vec{x}) \le (1+\eta)f(\vec{x})$. Therefore we get that

$$f(\vec{x}^* \vee (1+\eta)\vec{x}) - f((1+\eta)\vec{x}) \ge M - (1+\eta)f(\vec{x}) = \lambda\,.$$

This yields a contradiction. □

By the specification of the algorithm, the final solution is a good approximation. We show that it satisfies the packing constraints.

For the remainder of the analysis, we use j to index the iterations of the algorithm and we let $\vec{x}^{(j)}$ and $\vec{x}^{(j+1)}$ be the vector $\vec{x}$ at the beginning and end of iteration j, respectively. We let $\lambda^{(j)}, \vec{c}^{(j)}, \vec{m}^{(j)}, \vec{d}^{(j)}$ be the variables defined in iteration j.

Lemma 18. *The solution $\vec{x}$ returned by the algorithm satisfies $\|\mathbb{A}\vec{x}\|_\infty \le \mathrm{smax}_\eta(\mathbb{A}\vec{x}) \le 1-2\epsilon$.*

Proof. We show that the algorithm maintains the invariant that $\mathrm{smax}_\eta(\mathbb{A}\vec{x}) \le 1-\epsilon$. Since $\vec{x}^{(1)}$ is the initial vector defined on line 1 of Algorithm 2, we have $\mathrm{smax}_\eta(\mathbb{A}\vec{x}^{(1)}) \le \eta\ln m + \|\mathbb{A}\vec{x}^{(1)}\|_\infty \le 2\epsilon$. By Lemma 16, in every iteration j,

$$\begin{aligned}&\mathrm{smax}_\eta(\mathbb{A}\vec{x}^{(j+1)}) - \mathrm{smax}_\eta(\mathbb{A}\vec{x}^{(j)}) \\ &\le \frac{1}{M - (1+\eta)f(\vec{x}^{(j)})}\cdot\left(f(\vec{x}^{(j+1)}) - f(\vec{x}^{(j)})\right)\,.\end{aligned}$$

Let T be the final iteration. Summing up we get that

$$\mathrm{smax}_\eta(\mathbb{A}\vec{x}^{(T)}) \le 2\epsilon + \sum_{j=1}^{T-1}\frac{f(\vec{x}^{(j+1)}) - f(\vec{x}^{(j)})}{M - (1+\eta)f(\vec{x}^{(j)})}\,.$$

Define $g(\alpha) = f(\vec{x}^{(j)}) + \alpha(f(\vec{x}^{(j+1)}) - f(\vec{x}^{(j)}))$. Because $f(\vec{x}^{(j+1)}) \ge f(\vec{x}^{(j)})$, the function g is non-decreasing. We have

$$\begin{aligned}\int_0^1 \frac{g'(\alpha)}{M-(1+\eta)g(0)}d\alpha &\le \int_0^1 \frac{g'(\alpha)}{M-(1+\eta)g(\alpha)}d\alpha \\ &= \frac{1}{1+\eta}\ln\left(\frac{M-(1+\eta)g(0)}{M-(1+\eta)g(1)}\right)\,.\end{aligned}$$

Thus,

$$\mathrm{smax}_\eta(\mathbb{A}\vec{x}^{(T)}) \le 2\epsilon + \frac{1}{1+\eta}\cdot\ln\left(\frac{M-(1+\eta)f(\vec{x}^{(1)})}{M-(1+\eta)f(\vec{x}^{(T)})}\right)\,.$$

Using $f(\vec{x}^{(1)}) \geq 0$ and $M - (1+\eta)f(\vec{x}^{(T)}) \geq M - (1+\eta)M(1 - 1/\exp(1-10\epsilon)) \geq M(\exp(10\epsilon - 1) - \eta)$, due to the termination condition, we obtain

$$\begin{aligned}\operatorname{smax}_\eta(\mathbb{A}\vec{x}^{(T)}) &\leq 2\epsilon + \ln\left(\frac{1}{\exp(10\epsilon-1)-\eta}\right)\\ &\leq 2\epsilon + \ln\left(\frac{1}{(1+2\epsilon)\exp(8\epsilon-1)-\epsilon/4}\right)\\ &\leq 2\epsilon + 1 - 8\epsilon = 1 - 6\epsilon\,.\end{aligned}$$

Since the final iteration increases the softmax by at most ϵ, the lemma follows. □

Finally, we analyze the number of iterations performed before the algorithm terminates. We do this in two steps. First, we relate the value of a single coordinate to the update steps that have increased it. In the second step, we show that there must be a coordinate that has been updated sufficiently, so that it must have increased a lot after only a small number of iterations.

Formally, we first bound the total increment on each coordinate.

Lemma 19. *Consider coordinate i. If the final value of $\vec{x}_i$ is at most n/ϵ then $\sum_j \vec{m}_i^{(j)} = O(\log(n/\epsilon)/\eta)$.*

Proof. For every iteration j, we have

$$\vec{x}_i^{(j+1)} = \vec{x}_i^{(j)} + \vec{d}_i^{(j)} = \vec{x}_i^{(j)}(1 + \eta\vec{m}_i^{(j)}) \geq \vec{x}_i^{(j)}\exp(\eta\vec{m}_i^{(j)}/2)\,,$$

where we used $1 + z \geq \exp(z/2)$ for all $z \leq 1$. Therefore, letting T be the last iteration,

$$\vec{x}_i^{(T+1)} \geq \vec{x}_i^{(1)} \cdot \exp\left(\sum_{j=1}^{T} \eta\vec{m}_i^{(j)}/2\right).$$

Since the initial value of $\vec{x}_i$ is at least ϵ^2/n^2, and the final value of $\vec{x}_i$ is at most n/ϵ,

$$n^3/\epsilon^3 \geq \vec{x}_i^{(T+1)}/\vec{x}_i^{(1)} \geq \exp\left(\frac{\eta}{2}\sum_{j=1}^{T}\vec{m}_i^{(j)}\right),$$

which implies $\sum_{j=1}^{T}\vec{m}_i^{(j)} = O(\ln(n/\epsilon)/\eta)$. □

Finally, we bound the total number of iterations of the algorithm.

Lemma 20. *The number of iterations run by Algorithm 2 is at most $O\left(\frac{\log(n/\epsilon)}{\epsilon\eta}\right) = O\left(\frac{\log(n/\epsilon)\log(m)}{\epsilon^2}\right)$.*

Proof. First, we note that a simple analysis follows from partitioning the iterations into $O(\log(n/\epsilon))$ epochs, each of which corresponding to an interval where $\langle\vec{c},\vec{x}^*\rangle$ stays bounded within a constant multiplicative factor. Showing that for each of these phases, there exists a coordinate i for which $\sum_j \vec{m}_i^{(j)}$ is large enough will yield the result. Instead, we can obtain a refined bound on the number of iterations by partitioning the iterations into only two parts.

First, we notice that $\langle\vec{c},\vec{x}^*\rangle$ monotonically decreases over time, and $\lambda \in [M/3, M]$. Also, define $\vec{y} = \vec{c}/\lambda$. Using a similar argument to Lemma 17, we obtain that, for every iteration j,

$$\begin{aligned}\langle\vec{y}^{(j)},\vec{x}^*\rangle &= \frac{1}{\lambda^{(j)}}\langle\nabla f((1+\eta)\vec{x}^{(j)}),\vec{x}^*\rangle\\ &\geq \frac{1}{\lambda^{(j)}}\langle\nabla f((1+\eta)\vec{x}^{(j)}),\vec{x}^* \vee (1+\eta)\vec{x}^{(j)} - (1+\eta)\vec{x}^{(j)}\rangle\\ &\geq \frac{1}{\lambda^{(j)}} f(\vec{x}^* \vee (1+\eta)\vec{x}^{(j)}) - f((1+\eta)\vec{x}^{(j)})\\ &\geq \frac{1}{\lambda^{(j)}}(M - (1+\eta)f(\vec{x}^{(j)})\\ &= 1\,.\end{aligned}$$

Let j_2 be the last iteration of the algorithm, and let $v_2 = \langle\vec{c}^{(j_2)},\vec{x}^*\rangle$. We divide the iterations into two parts: let T_2 be the iterations j where $v_2 \leq \langle\vec{c}^{(j)},\vec{x}^*\rangle < 9v_2$, and T_1 be the iterations where $\langle\vec{c}^{(j)},\vec{x}^*\rangle \geq 9v_2$.

First we bound the number of iterations in T_2. Consider an iteration j in T_2. By definition, we have $\langle\vec{y}^{(j)},\vec{x}^*\rangle \in [v_2/\lambda^{(1)}, 27v_2/\lambda^{(1)}]$ so there exists $\alpha \leq 1$ so that $\langle\alpha\vec{y}^{(j)},\vec{x}^*\rangle \in [1, 27]$ for all iterations $j \in T_2$.

We also have $\langle\nabla\operatorname{smax}(\mathbb{A}\vec{x}^{(j)}), \mathbb{A}\vec{x}^*\rangle \leq \|\nabla\operatorname{smax}_\eta(\mathbb{A}\vec{x}^{(j)})\|_1\|\mathbb{A}\vec{x}^*\|_\infty \leq 1 - \epsilon$, which also gives us that

$$\left\langle\mathbb{D}(\alpha\vec{y}^{(j)})^+\mathbb{A}^\top\nabla\operatorname{smax}(\mathbb{A}\vec{x}^{(j)}), \mathbb{D}(\alpha\vec{y}^{(j)})\vec{x}^*\right\rangle \leq 1 - \epsilon\,.$$

Combining this with $\langle\alpha\vec{y}^{(j)},\vec{x}^*\rangle \in [1, 27]$, we obtain that

$$\left\langle\vec{1} - \mathbb{D}(\alpha\vec{y}^{(j)})^+\mathbb{A}^\top\nabla\operatorname{smax}(\mathbb{A}\vec{x}^{(j)}), \mathbb{D}(\alpha\vec{y}^{(j)})\vec{x}^*\right\rangle \geq \epsilon\,.$$

Therefore adding up across all iterations in T_2, and using $\vec{y}^{(j)} \leq 3\vec{y}^{(j_0)}$, where j_0 is the first iteration in T_2, we have

$$\sum_{j\in T_2}\left\langle\vec{1} - \mathbb{D}(\alpha\vec{y}^{(j)})^+\mathbb{A}^\top\nabla\operatorname{smax}(\mathbb{A}\vec{x}^{(j)}), \mathbb{D}(\alpha\vec{y}^{(j_0)})\vec{x}^*\right\rangle \geq \epsilon|T_2|/3\,.$$

Because $\|\mathbb{D}(\alpha\vec{y}^{(j_0)})\vec{x}^*\|_1 \leq 27$, by averaging, there exists a coordinate i such that

$$\sum_{j\in T_2}\left(\left(1 - \frac{(\mathbb{A}^\top\nabla\operatorname{smax}(\mathbb{A}\vec{x}^{(j)}))_i}{\alpha\vec{y}_i^{(j)}}\right)\vee 0\right) \geq \epsilon|T_2|/81\,.$$

Using the fact that $\alpha \leq 1$ and the definition of $\vec{y}$, we see that this also gives us that

$$\sum_{j\in T_2}\vec{m}_i^{(j)} \geq \epsilon|T_2|/81\,,$$

so, by Lemma 19, we have $|T_2| = O(\log(n/\epsilon)/(\epsilon\eta))$.

Next we bound the number of iterations in T_1. For any iteration $j \in T_1$, we have

$$\langle\vec{y}^{(j)},\vec{x}^*\rangle \geq 9v_2/\lambda^{(j)} \geq 9v_2/\lambda^{(1)} \geq 3\,.$$

Let j_1 be the last iteration in T_1. Let $\alpha = \frac{3}{\langle\vec{y}^{(j_1)},\vec{x}^*\rangle} \leq 1$. Similarly to before, we have

$$\left\langle\mathbb{D}(\alpha\vec{y}^{(j)})^+\mathbb{A}^\top\nabla\operatorname{smax}(\mathbb{A}\vec{x}^{(j)}), \mathbb{D}\left(\frac{\alpha}{3}\vec{y}^{(j)}\right)\vec{x}^*\right\rangle \leq 1 - \epsilon\,,$$

where we use $\vec{y}^{(j)} = \frac{1}{\lambda^{(j)}}\vec{c}^{(j)} \geq \frac{1}{\lambda^{(j)}}\vec{c}^{(j_1)} = \frac{\lambda^{(j_1)}}{\lambda^{(j)}}\vec{y}^{(j_1)} \geq \frac{1}{3}\vec{y}^{(j_1)}$. Thus for our specific choice of α we obtain:

$$\left\langle\vec{1} - \mathbb{D}(\alpha\vec{y}^{(j)})^+\mathbb{A}^\top\nabla\operatorname{smax}(\mathbb{A}\vec{x}^{(j)}), \mathbb{D}\left(\frac{\alpha}{3}\vec{y}^{(j_1)}\right)\vec{x}^*\right\rangle \geq \epsilon\,.$$

So adding up across all iterations in T_1, and using $\|\mathbb{D}(\alpha\vec{y}^{(j_1)})\vec{x}^*\|_1 = 3$, by averaging, there exists a coordinate i such that

$$\sum_{j\in T_1}\left(\left(1-\frac{(\mathbb{A}^\top\nabla\text{smax}(\mathbb{A}\vec{x}^{(j)}))_i}{\alpha\vec{y}_i^{(j)}}\right)\vee 0\right)\geq\epsilon|T_1|\,.$$

Just like before, by Lemma 19, this gives us that $|T_1| = O(\log(n/\epsilon)/(\epsilon\eta))$.

□

5 NON-MONOTONE MAXIMIZATION WITH PACKING CONSTRAINTS

Our algorithm for non-monotone DR-submodular maximization is shown in Algorithm 3. We obtain the value M by guessing like in the monotone case. Unlike the monotone case, we now include the constraints $\vec{x}\leq(1-\epsilon)\vec{1}$ into the matrix $\mathbb{A}$, i.e., we solve the problem $\max\{\vec{x}\in\mathbb{R}_+^n\colon \mathbb{A}\vec{x}\leq(1-\epsilon)\vec{1}\}$ where the constraints $\mathbb{A}\vec{x}\leq(1-\epsilon)\vec{1}$ include the constraints $\vec{x}\leq(1-\epsilon)\vec{1}$. For simplicity, we let m denote the number of rows of this enlarged matrix $\mathbb{A}$, i.e., $m = m' + n$ where m' is the original number of packing constraints. Due to space constraints, we omit the proofs. They can be found in the full version of our paper [16].

Algorithm 3 Algorithm for $\max_{\vec{x}\in\mathbb{R}_+^n\colon \mathbb{A}\vec{x}\leq(1-\epsilon)\vec{1}} f(\vec{x})$, where f is a non-negative DR-submodular function and $\mathbb{A}\in\mathbb{R}_+^{m\times n}$. The constraint $\mathbb{A}\vec{x}\leq(1-\epsilon)\vec{1}$ includes the constraints $\vec{x}\leq(1-\epsilon)\vec{1}$.

$\eta\leftarrow\frac{\epsilon}{2\ln m}$
M is an approximate optimal solution value: $M\leq f(\vec{x}^*)\leq(1+\epsilon)M$, where $\vec{x}^*\in\text{argmax}_{\vec{x}\colon \mathbb{A}\vec{x}\leq(1-\epsilon)\vec{1}} f(\vec{x})$.
$\vec{x}_i\leftarrow\frac{\epsilon}{n\|\mathbb{A}_{:i}\|_\infty},\quad\forall i\in[n]$
$\vec{z}\leftarrow\vec{x}$
$t\leftarrow\text{smax}_\eta(\mathbb{A}\vec{z})$
while $f(\vec{x})\leq\exp(-1-10\epsilon)M$ **do**
  $\lambda\leftarrow M\cdot(e^{-t}-2\epsilon)-f(\vec{x})$
  $\vec{c}_i\leftarrow(1-\vec{x}_i)\nabla_i f((1+\eta)\vec{x})\vee 0\quad\forall i\in[n]$
  $\vec{m}_i\leftarrow\left(1-\lambda\cdot\frac{(\mathbb{A}^\top\nabla\text{smax}(\mathbb{A}\vec{z}))_i}{\vec{c}_i}\right)\vee 0$ for all i with $\vec{c}_i\neq 0$, and $\vec{m}_i = 0$ if $\vec{c}_i = 0$
  $\vec{d}\leftarrow\eta\vec{x}\circ\vec{m}$
  $\vec{x}\leftarrow\vec{x}+\vec{d}\circ(\vec{1}-\vec{x})$
  $\vec{z}\leftarrow\vec{z}+\vec{d}$
  $t\leftarrow\text{smax}_\eta(\mathbb{A}\vec{z})$
end while

Lemma 21. *We have*

$$\frac{f(\vec{x}+\vec{d}\circ(\vec{1}-\vec{x}))-f(\vec{x})}{\text{smax}_\eta(\mathbb{A}(\vec{z}+\vec{d}))-\text{smax}_\eta(\mathbb{A}\vec{z})}\geq\lambda.$$

Lemma 22. *Let t and t' be the values of t at the beginning and the end of an iteration. Assume that $t'\leq 1$. Let $\vec{x}$ and $\vec{x}'$ be the values of $\vec{x}$ at the beginning and the end of the same iteration. We have*

$$e^{t'}\cdot f(\vec{x}')\geq(1-2e\epsilon)(t'-t)M+e^t\cdot f(\vec{x})$$

Lemma 23. *We have the invariant that $\|x\|_\infty\leq(1+\epsilon)(1-e^{-t})$.*

Lemma 24. *In every iteration, we have $\vec{d}\neq\vec{0}$.*

By the design of the algorithm, the final solution is a good approximation. The following lemma shows that it satisfies the packing constraints.

Lemma 25. *The solution $\vec{x}$ returned by the algorithm satisfies $\|\mathbb{A}\vec{x}\|_\infty\leq\text{smax}_\eta(\mathbb{A}\vec{x})\leq 1-2\epsilon$.*

Finally, we can bound the total number of iterations:

Lemma 26. *The number of iterations is at most $O\left(\frac{\log(n/\epsilon)\log(1/\epsilon)}{\epsilon\eta}\right) = O\left(\frac{\log(n/\epsilon)\log(1/\epsilon)\log(m)}{\epsilon^2}\right)$.*

6 NON-MONOTONE MAXIMIZATION WITH A MATROID CONSTRAINT

In this section, we consider the problem of maximizing a non-monotone DR-submodular function subject to a polymatroid constraint. The algorithm and analysis are an extension of the algorithm and analysis for monotone functions from Section 3. The key modification to the algorithm is the multiplication by $\vec{1}-\vec{z}$ to dampen the growth of the solution that we borrow from the measured continuous greedy algorithm [19]. For complete proofs, we refer the reader to the full version of our paper [16].

Algorithm 4 Algorithm for non-monotone maximization subject to a polymatroid constraint.

M is an approximate optimal solution value: $M\leq f(\vec{x}^*)\leq(1+\epsilon)M$, where $\vec{x}^*\in\text{argmax}_{\vec{x}\in\mathbf{P}} f(\vec{x})$, D is chosen such that $\|\nabla f(\vec{x})\|_\infty\leq MD$.
$\vec{z}\leftarrow 0$
for $j\leftarrow 0$ to $1/\epsilon-1$ **do**
  $\vec{x}^{(0)}\leftarrow\frac{\epsilon^2}{nD}\vec{1}$
  $t\leftarrow 0$
  Let $g(\vec{x}) = f((\vec{1}-\vec{z})\circ\vec{x}+\vec{z})$
  while $g(\vec{x}^{(t)})-g(\vec{x}^{(0)})\leq\epsilon\left((\frac{1}{1+\epsilon})^j-10\epsilon)M-g(\vec{x}^{(0)})\right)$ **do**
    $\vec{c}_i\leftarrow\nabla_i g((1+\epsilon)\vec{x}^{(t)})$
    $= (1-\vec{z}_i)\nabla_i f((\vec{1}-\vec{z})\circ(1+\epsilon)\vec{x}^{(t)}+\vec{z})$
    Let $T(\vec{x})$ for $\vec{x}\in\frac{\epsilon}{1+\epsilon}\mathbf{P}$ be the *maximal* set S such that $\vec{x}(S) = \frac{\epsilon}{1+\epsilon}r(S)$.
    Let $v_1 = \max_{i\notin T(\vec{x}^{(t)})}\vec{c}_i$ and v_2 be the maximum power of $1+\epsilon$ such that $v_2\leq v_1$.
    $\vec{y}\leftarrow 0$
    for i from 1 to n **do**
      if $\vec{c}_i\geq v_2$ **then**
        Let $\vec{y}_i$ be the maximum value such that $\vec{y}_i\leq\epsilon\vec{x}_i^{(t)}$ and $(1+\epsilon)(\vec{x}^{(t)}+\vec{y})\in\epsilon\mathbf{P}$.
      end if
    end for
    $\vec{x}^{(t+1)}\leftarrow\vec{x}^{(t)}+\vec{y}$
    $t\leftarrow t+1$
  end while
  $\vec{z}\leftarrow\vec{z}+(\vec{1}-\vec{z})\circ\vec{x}^{(t)}$
end for
return $\vec{z}$

Analysis of the approximation guarantee. We show that the algorithm achieves a $1/e-O(\epsilon)$ approximation guarantee. We

consider each iteration of the algorithm and we analyze the increase in value when updating $\vec{x}^{(t)}$ to $\vec{x}^{(t+1)}$. Similarly to the case of non-monotone maximization, we use Lemma 7 to show that we can define a sequence of vectors $\vec{o}^{(t)}$ based on $\vec{x}^{(t)}$ and the optimal solution that allows us to relate the gain of the algorithm to the optimum value. Use use the same properties of the vector $\vec{o}^{(t)}$ as those given in Section 3.

For the analysis, we employ a simple bound on $\|\vec{z}^{(j)}\|_\infty$, which is used to bound the approximation.

Lemma 27. $\|\vec{z}^{(j)}\|_\infty \leq 1 - (1 - \epsilon/(1+\epsilon))^j$.

We now relate the gain in our solution in every iteration to the change in $\vec{o}$. The proof is similar to the one in the monotone case.

Lemma 28 (cf Lemma 9). *We have*

$$g(\vec{x}^{(t+1)}) - g(\vec{x}^{(t)}) \geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} \left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}) \vee \vec{0}, (1-\vec{z}) \circ (\vec{o}^{(t)} - \vec{o}^{(t+1)}) \right\rangle .$$

By repeatedly applying Lemma 28, we obtain the following lemma.

Lemma 29 (cf Lemma 10). *We have*

$$g(\vec{x}^{(t+1)}) - g(\vec{x}^{(0)}) \geq \frac{\epsilon(1-\epsilon)}{1+\epsilon} \left\langle \nabla g((1+\epsilon)\vec{x}^{(t)}) \vee \vec{0}, \vec{o}^{(0)} - \vec{o}^{(t+1)} \right\rangle .$$

This implies that every iteration of the while loop increases at least one coordinate, and thus the while loop eventually terminates.

Lemma 30. *In every iteration t, we have $T(\vec{x}^{(t)}) \neq V$, i.e., some coordinate increases in each iteration.*

Thus the algorithm terminates. Finally, we show that the solution returned is a $1/e - O(\epsilon)$ approximation:

Lemma 31. *The solution $\vec{z}$ returned by Algorithm 4 is feasible and it satisfies $f(\vec{z}) \geq (1/e - O(\epsilon))M \geq (1/e - O(\epsilon))f(\vec{x}^*)$.*

Analysis of the number of iterations. The following lemma upper bounds the total number of iterations of Algorithm 4, and thus the number of rounds of adaptivity.

Lemma 32. *The total number of iterations and rounds of adaptivity is $O(\log^2 n/\epsilon^3)$.*

ACKNOWLEDGMENTS

AE was partially supported by NSF CAREER grant CCF-1750333 and NSF grant CCF-1718342. HN was partially supported by NSF CAREER grant CCF-1750716. AV was partially supported by NSF grant CCF-1718342.

REFERENCES

[1] Zeyuan Allen-Zhu and Lorenzo Orecchia. 2015. Using optimization to break the epsilon barrier: A faster and simpler width-independent algorithm for solving positive linear programs in parallel. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 1439–1456.

[2] Francis Bach. 2016. Submodular functions: from discrete to continuous domains. *Mathematical Programming* (2016), 1–41.

[3] Eric Balkanski, Adam Breuer, and Yaron Singer. 2018. Non-monotone Submodular Maximization in Exponentially Fewer Iterations. *arXiv preprint arXiv:1807.11462* (2018).

[4] Eric Balkanski, Aviad Rubinstein, and Yaron Singer. 2018. An Exponential Speedup in Parallel Running Time for Submodular Maximization without Loss in Approximation. *CoRR* abs/1804.06355 (2018).

[5] Eric Balkanski and Yaron Singer. 2018. The Adaptive Complexity of Maximizing a Submodular Function. In *ACM Symposium on Theory of Computing (STOC)*.

[6] Rafael D.P. Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. 2015. The Power of Randomization: Distributed Submodular Maximization on Massive Datasets. In *International Conference on Machine Learning (ICML)*.

[7] Rafael da Ponte Barbosa, Alina Ene, Huy L Nguyen, and Justin Ward. 2016. A new framework for distributed submodular maximization. In *IEEE Foundations of Computer Science (FOCS)*. 645–654.

[8] An Bian, Joachim M Buhmann, and Andreas Krause. 2018. Optimal DR-Submodular Maximization and Applications to Provable Mean Field Inference. *arXiv preprint arXiv:1805.07482* (2018).

[9] An Bian, Kfir Levy, Andreas Krause, and Joachim M Buhmann. 2017. Continuous dr-submodular maximization: Structure and algorithms. In *Advances in Neural Information Processing Systems*. 486–496.

[10] Andrew An Bian, Baharan Mirzasoleiman, Joachim M Buhmann, and Andreas Krause. 2016. Guaranteed non-convex optimization: Submodular maximization over continuous domains. *arXiv preprint arXiv:1606.05615* (2016).

[11] Niv Buchbinder and Moran Feldman. 2016. Constrained submodular maximization via a non-symmetric technique. *arXiv preprint arXiv:1611.03253* (2016).

[12] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. 2011. Maximizing a Submodular Set Function Subject to a Matroid Constraint. *SIAM J. Comput.* 40, 6 (2011), 1740–1766.

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

[14] Chandra Chekuri and Kent Quanrud. 2018. Submodular Function Maximization in Parallel via the Multilinear Relaxation. *arXiv preprint arXiv:1807.08678* (2018).

[15] Alina Ene and Huy L Nguyen. 2018. Submodular Maximization with Nearly-optimal Approximation and Adaptivity in Nearly-linear Time. *arXiv preprint arXiv:1804.05379* (2018).

[16] Alina Ene, Huy L Nguyen, and Adrian Vladu. 2018. Submodular maximization with matroid and packing constraints in parallel. *arXiv preprint arXiv:1808.09987* (2018).

[17] Alessandro Epasto, Vahab Mirrokni, and Morteza Zadimoghaddam. 2017. Bicriteria Distributed Submodular Maximization in a Few Rounds. In *PACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 25–33.

[18] Matthew Fahrbach, Vahab Mirrokni, and Morteza Zadimoghaddam. 2018. Submodular Maximization with Optimal Approximation, Adaptivity and Query Complexity. *arXiv preprint arXiv:1807.07889* (2018).

[19] Moran Feldman, Joseph Naor, and Roy Schwartz. 2011. A Unified Continuous Greedy Algorithm for Submodular Maximization. In *IEEE Foundations of Computer Science (FOCS)*. https://doi.org/10.1109/FOCS.2011.46

[20] András Frank. 2011. *Connections in combinatorial optimization*. Vol. 38. OUP Oxford.

[21] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. 2013. Fast Greedy Algorithms in Mapreduce and Streaming. In *PACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 1–10.

[22] Michael Luby and Noam Nisan. 1993. A parallel approximation algorithm for positive linear programming. In *ACM Symposium on Theory of Computing (STOC)*. ACM, 448–457.

[23] Michael W Mahoney, Satish Rao, Di Wang, and Peng Zhang. 2016. Approximating the Solution to Mixed Packing and Covering LPs in Parallel O (epsilon^{-3}) Time. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 55. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[24] Vahab Mirrokni and Morteza Zadimoghaddam. 2015. Randomized composable core-sets for distributed submodular maximization. In *ACM Symposium on Theory of Computing (STOC)*.

[25] Baharan Mirzasoleiman, Amin Karbasi, Ashwinkumar Badanidiyuru, and Andreas Krause. 2015. Distributed submodular cover: Succinctly summarizing massive data. In *Advances in Neural Information Processing Systems*. 2881–2889.

[26] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. 2013. Distributed Submodular Maximization: Identifying Representative Elements in Massive Data. In *Advances in Neural Information Processing Systems (NIPS)*. 2049–2057.

[27] Arkadi Nemirovski. 1994. On parallel complexity of nonsmooth convex optimization. *J. Complexity* 10, 4 (1994), 451–463.

[28] Rad Niazadeh, Tim Roughgarden, and Joshua R Wang. 2018. Optimal Algorithms for Continuous Non-monotone Submodular and DR-Submodular Maximization. *arXiv preprint arXiv:1805.09480* (2018).

[29] Tasuku Soma and Yuichi Yoshida. 2017. Non-Monotone DR-Submodular Function Maximization.. In *AAAI*, Vol. 17. 898–904.

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

[31] Neal E Young. 2001. Sequential and parallel algorithms for mixed packing and covering. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*. IEEE, 538–546.