Non-Monotone DR-Submodular Function Maximization

Tasuku Soma  
The University of Tokyo  
tasuku.soma@mist.i.u-tokyo.ac.jp

Yuichi Yoshida  
National Institute of Informatics, and  
Preferred Infrastructure, Inc.  
yyoshida@nii.ac.jp

Abstract
We consider non-monotone DR-submodular function maximization, where DR-submodularity (diminishing return submodularity) is an extension of submodularity for functions over the integer lattice based on the concept of the diminishing return property. Maximizing non-monotone DR-submodular functions has many applications in machine learning that cannot be captured by submodular set functions. In this paper, we present a $\frac{1}{1+\epsilon}$-approximation algorithm with a running time of roughly $O(\log^2 B)$, where $n$ is the size of the ground set, $B$ is the maximum value of a coordinate, and $\epsilon > 0$ is a parameter. The approximation ratio is almost tight and the dependency of running time on $B$ is exponentially smaller than the naive greedy algorithm. Experiments on synthetic and real-world datasets demonstrate that our algorithm outputs almost the best solution compared to other baseline algorithms, whereas its running time is several orders of magnitude faster.

Introduction
Submodular functions have played a key role in various tasks in machine learning, statistics, social science, and economics. A set function $f : 2^E \rightarrow \mathbb{R}$ with a ground set $E$ is submodular if

$$f(X \cup \{e\}) - f(X) \geq f(Y \cup \{e\}) - f(Y)$$

for arbitrary sets $X, Y \subseteq E$ with $X \subseteq Y$, and an element $e \in E \setminus Y$. The importance and usefulness of submodularity in these areas are due to the fact that submodular functions naturally capture the diminishing return property. Various important functions in these areas such as the entropy function, coverage function, and utility functions satisfy this property. See, e.g., (Krause and Golovin 2014; Fujishige 2005).

Recently, maximizing (non-monotone) submodular functions has attracted particular interest in the machine learning community. In contrast to minimizing submodular functions, which can be done in polynomial time, maximizing submodular functions is NP-hard in general. However, we can achieve a constant factor approximation for various settings. Notably, (Buchbinder et al. 2012) presented a very elegant double greedy algorithm for (unconstrained) submodular function maximization, which was the first algorithm achieving $\frac{1}{2}$-approximation, and this approximation ratio is tight (Feige, Mirrokni, and Vondrak 2011). Applications of non-monotone submodular function maximization include efficient sensor placement (Krause, Singh, and Guestrin 2008), privacy in online services (Krause and Horvitz 2008), and maximum entropy sampling (Ko, Lee, and Queyranne 1995).

The models and applications mentioned so far are built upon submodular set functions. Although set functions are fairly powerful for describing problems such as variable selection, we sometimes face difficult situations that cannot be cast with set functions. For example, in the budget allocation problem (Alon, Ganzu, and Tennenholtz 2012), we would like to decide how much budget should be set aside for each ad source, rather than whether we use the ad source or not. A similar issue arises when we consider models allowing multiple choices of an element in the ground set.

To deal with such situations, several generalizations of submodularity have been proposed. (Soma et al. 2014) devised a general framework for maximizing monotone submodular functions on the integer lattice, and showed that the budget allocation problem and its variant fall into this framework. In their framework, functions are defined over the integer lattice $\mathbb{Z}^E_x$ and therefore effectively represent discrete allocations of budget. Regarding the original motivation for the diminishing return property, one can naturally consider its generalization to the integer lattice: a function $f : \mathbb{Z}^E_x \rightarrow \mathbb{R}$ satisfying

$$f(x + \chi_e) - f(x) \geq f(y + \chi_e) - f(y)$$

for $x \leq y$ and $e \in E$, where $\chi_e \in \mathbb{R}^E$ is the vector with $\chi_e(e) = 1$ and $\chi_e(a) = 0$ for every $a \neq e$. Such functions are called diminishing return submodular (DR-submodular) functions (Soma and Yoshida 2015) or coordinate-wise concave submodular functions (Milgrom and Strulovici 2009). DR-submodular functions have found various applications in generalized sensor placement (Soma and Yoshida 2015) and (a natural special case of) the budget allocation problem (Soma et al. 2014).

As a related notion, a function is said to be lattice submodular if

$$f(x) + f(y) \geq f(x \lor y) + f(x \land y)$$

for arbitrary $x$ and $y$, where $\lor$ and $\land$ are coordinate-wise max and min, respectively. Note that DR-submodularity is
stronger than lattice submodularity in general (see, e.g., (Soma et al. 2014)). Nevertheless, we consider the DR-submodularity to be a “natural definition” of submodularity, at least for the applications mentioned so far, because the diminishing return property is crucial in these real-world scenarios.

Our contributions

We design a novel polynomial-time approximation algorithm for maximizing (non-monotone) DR-submodular functions. More precisely, we consider the optimization problem

\[
\text{maximize } f(x) \\
\text{subject to } 0 \leq x \leq B,
\]

where \( f : \mathbb{Z}_+^E \rightarrow \mathbb{R}_+ \) is a non-negative DR-submodular function, \( 0 \) is the zero vector, and \( B \in \mathbb{Z}_+^E \) is a vector representing the maximum values for each coordinate. When \( B \) is the all-ones vector, this is equivalent to the original (unconstrained) submodular function maximization. We assume that \( f \) is given as an evaluation oracle; when we specify \( x \in \mathbb{Z}_+^E \), the oracle returns the value of \( f(x) \).

Our algorithm achieves \( \frac{1}{2+\epsilon} \)-approximation for any constant \( \epsilon > 0 \) in \( O\left( \frac{|E|}{\epsilon} \log(\frac{\epsilon}{\delta}) \log B \cdot (\theta + \log B) \right) \) time, where \( \delta \) and \( \Delta \) are the minimum positive marginal gain and maximum positive values, respectively, of \( f, B = \|B\|_{\infty} := \max_{e \in E} B(e), \) and \( \theta \) is the running time of evaluating the (oracle for) \( f \). To our knowledge, this is the first polynomial-time algorithm achieving (roughly) \( \frac{1}{2} \)-approximation.

We also conduct numerical experiments on the revenue maximization problem using real-world networks. The experimental results show that the solution quality of our algorithm is comparable to other algorithms. Furthermore, our algorithm runs several orders of magnitude faster than other algorithms when \( B \) is large.

DR-submodularity is necessary for obtaining polynomial-time algorithms with a meaningful approximation guarantee; if \( f \) is only lattice submodular, then we cannot obtain constant-approximation in polynomial time. To see this, it suffices to observe that an arbitrary univariate function is lattice submodular, and therefore finding an (approximate) maximum value must invoke \( O(B) \) queries. We note that representing an integer \( B \) requires \( \lceil \log_2 B \rceil \) bits. Hence, the running time of \( O(B) \) is pseudopolynomial rather than polynomial.

Fast simulation of the double greedy algorithm

Naturally, one can reduce the problem (1) to maximization of a submodular set function by simply duplicating each element \( e \) in the ground set into \( B(e) \) distinct copies and defining a set function over the set of all the copies. One can then run the double greedy algorithm (Buchbinder et al. 2012) to obtain \( \frac{1}{2} \)-approximation. This reduction is simple but has one large drawback; the size of the new ground set is \( \sum_{e \in E} B(e) \), a pseudopolynomial in \( B \). Therefore, this naive double greedy algorithm does not scale to a situation where \( B \) is large.

For scalability, we need an additional trick that reduces the pseudo-polynomial running time to a polynomial one. In monotone submodular function maximization on the integer lattice, (Soma and Yoshida 2015; 2016) provide such a speedup trick, which effectively combines the decreasing threshold technique (Badanidiyuru and Vondrak 2014) with binary search. However, a similar technique does not apply to our setting, because the double greedy algorithm works differently from (single) greedy algorithms for monotone submodular function maximization. The double greedy algorithm examines each element in a fixed order and marginal gains are used to decide whether to include the element or not. In contrast, the greedy algorithm chooses each element in decreasing order of marginal gains, and this property is crucial for the decreasing threshold technique.

We resolve this issue by splitting the set of all marginal gains into polynomially many small intervals. For each interval, we approximately execute multiple steps of the double greedy algorithm at once, as long as the marginal gains remain in the interval. Because the marginal gains do not change (much) within the interval, this simulation can be done with polynomially many queries and polynomial-time overhead. To our knowledge, this speedup technique is not known in the literature and is therefore of more general interest.

Very recently, (Ene and Nguyen 2016) pointed out that a DR-submodular function \( \tilde{f} : \{0, 1, \ldots, B\}^E \rightarrow \mathbb{R}_+ \) can be expressed as a submodular set function \( g \) over a polynomial-sized ground set, which turns out to be \( E \times \{0, 1, \ldots, k - 1\} \), where \( k = \lceil \log_2 (B + 1) \rceil \). Their idea is representing \( x(e) \) in binary form for each \( e \in E \), and bits in the binary representations form the new ground set. We may want to apply the double greedy algorithm to \( g \) in order to get a polynomial-time approximation algorithm. However, this strategy has the following two drawbacks: (i) The value of \( g(E \times \{0, 1, \ldots, k - 1\}) \) is defined as \( f(x) \), where \( x(e) = 2^e - 1 \) for every \( e \in E \). This means that we have to extend the domain of \( f \). (ii) More crucially, the double greedy algorithm on \( g \) may return a large set such as \( E \times \{0, 1, \ldots, k - 1\} \) whose corresponding vector \( x \in \mathbb{Z}_+^E \) may violate the constraint \( x \leq B \). Although we can resolve these issues by introducing a knapsack constraint, it is not a practical solution because existing algorithms for knapsack constraints (Lee et al. 2009; Chekuri, Vondrák, and Zenklusen 2014) are slow and have worse approximation ratios than 1/2.

Notations

For an integer \( n \in \mathbb{N} \), \([n]\) denotes the set \( \{1, \ldots, n\} \). For vectors \( x, y \in \mathbb{Z}_+^E \), we define \( f(x \mid y) := f(x + y) - f(y) \). The \( \ell_1 \)-norm and \( \ell_{\infty} \)-norm of a vector \( x \in \mathbb{Z}_+^E \) are defined as \( \|x\|_1 := \sum_{e \in E} |x(e)| \) and \( \|x\|_{\infty} := \max_{e \in E} |x(e)| \), respectively.

Related work

As mentioned above, there have been many efforts to maximize submodular functions on the integer lattice. Perhaps the work most related to our interest is (Gottschalk and Peis 2015), in which the authors considered maximizing

899
lattice submodular functions over the bounded integer lattice and designed \( \frac{1}{2} \)-approximation pseudopolynomial-time algorithm. Their algorithm was also based on the double greedy algorithm, but does not include a speeding up technique, as proposed in this paper.

In addition there are several studies on the constrained maximization of submodular functions (Feige, Mirrokni, and Vondrak 2011; Buchbinder et al. 2014; Buchbinder and Feldman 2016), although we focus on the unconstrained case. Many algorithms for maximizing submodular functions are randomized, but a very recent work (Buchbinder, Feldman 2015) considered maximizing non-monotone submodular functions (Feige, Mirrokni, and Vondrak 2016), which runs in pseudo-polynomial time. Then, we show that the input function \( f \) and \( f(-x \mid y) \); good approximations to them are sufficient to achieve an approximation guarantee close to \( \frac{1}{2} \). To exploit this observation, we first design an algorithm that outputs (sketches of) approximations to the functions \( g(b) := f(x_e \mid x + b \chi_e) \) and \( h(b) := f(-x_e \mid y - b \chi_e) \), then \( \phi(b) \). Note that \( g \) and \( h \) are non-increasing functions in \( b \) because of the DR-submodularity of \( f \).

To illustrate this idea, we let us consider a non-increasing function \( \phi : \{0,1,\ldots,B-1\} \to \mathbb{R} \) and suppose that \( \phi \) is non-negative (\( \phi \) is either \( g \) or \( h \) later on). Let \( \delta \) and \( \Delta \) be the minimum and the maximum positive values of \( \phi \), respectively. Then, for each \( \delta \leq \tau \leq \Delta \) of the form \( \delta(1+\epsilon)^k \), we find the minimum \( b_\tau \), such that \( \phi(b_\tau) \leq \delta \) (we regard \( \phi(B) = -\infty \)). From the non-increasing property of \( \phi \), we then have \( \phi(b) \geq \tau \) for any \( b \leq b_\tau \). Using the set of pairs \( \{\tau, b_\tau\} \), we can obtain a good approximation to \( \phi \). The details are provided in Algorithm 2.

Algorithm 1: Pseudopolynomial-time algorithm

| Input: | \( f : \mathbb{Z}_+^E \to \mathbb{R}_+ \), \( B \in \mathbb{Z}_+^E \). |
| Assumption: | \( f \) is DR-submodular. |
| Algorithm 1 | 1: \( x \leftarrow 0 \), \( y \leftarrow B \). |
| Algorithm 1 | 2: for \( e \in E \) do |
| Algorithm 1 | 3: \( x(e) \leftarrow y(e) \) do |
| Algorithm 1 | 4: \( \alpha \leftarrow f(x_e \mid x) \) and \( \beta \leftarrow f(-x_e \mid y) \). |
| Algorithm 1 | 5: if \( \beta < 0 \) then |
| Algorithm 1 | 6: \( x(e) \leftarrow x(e) + 1 \). |
| Algorithm 1 | 7: else if \( \alpha < 0 \) then |
| Algorithm 1 | 8: \( y(e) \leftarrow y(e) + 1 \). |
| Algorithm 1 | 9: else |
| Algorithm 1 | 10: \( x(e) \leftarrow x(e) + 1 \) with probability \( \frac{\alpha}{\alpha + \beta} \) and |
| Algorithm 1 | 11: \( y(e) \leftarrow y(e) + 1 \) with the complement probability \( \frac{\beta}{\alpha + \beta} \). |
| Algorithm 1 | 12: end if |
| Algorithm 1 | 13: end while |
| Algorithm 1 | 14: return \( x \). |

Theorem 1. Algorithm 1 is a \( \frac{1}{2} \)-approximation algorithm for (1) with time complexity \( O(|B|_1 \cdot \theta + |B|_1) \), where \( \theta \) is the running time of evaluating \( f \).

We omit the proof as it is a simple modification of the analysis of the original algorithm.

Algorithm with polynomially many oracle calls

In this section, we present an algorithm with a polynomial number of oracle calls.

Our strategy is to simulate Algorithm 1 without evaluating the input function \( f \) many times. A key observation is that, at Line 4 of Algorithm 1, we do not need to know the exact value of \( f(x_e \mid x) \) and \( f(-x_e \mid y) \); good approximations to them are sufficient to achieve an approximation guarantee close to \( \frac{1}{2} \). To exploit this observation, we first design an algorithm that outputs (sketches of) approximations to the functions \( g(b) := f(x_e \mid x + b \chi_e) \) and \( h(b) := f(-x_e \mid y - b \chi_e) \), then \( \phi(b) \). Note that \( g \) and \( h \) are non-increasing functions in \( b \) because of the DR-submodularity of \( f \).

To illustrate this idea, we let us consider a non-increasing function \( \phi : \{0,1,\ldots,B-1\} \to \mathbb{R} \) and suppose that \( \phi \) is non-negative (\( \phi \) is either \( g \) or \( h \) later on). Let \( \delta \) and \( \Delta \) be the minimum and the maximum positive values of \( \phi \), respectively. Then, for each \( \delta \leq \tau \leq \Delta \) of the form \( \delta(1+\epsilon)^k \), we find the minimum \( b_\tau \), such that \( \phi(b_\tau) \leq \delta \) (we regard \( \phi(B) = -\infty \)). From the non-increasing property of \( \phi \), we then have \( \phi(b) \geq \tau \) for any \( b \leq b_\tau \). Using the set of pairs \( \{\tau, b_\tau\} \), we can obtain a good approximation to \( \phi \). The details are provided in Algorithm 2.

Lemma 2. For any \( \phi : \{0,1,\ldots,B-1\} \to \mathbb{R} \) and \( \epsilon > 0 \), Algorithm 2 outputs a set of pairs \( \{(b_\tau, \tau)\} \), from which, for any \( b \in \{0,1,\ldots,B-1\} \), we can reconstruct a value \( v \) in \( O(\log B) \) time such that \( v \leq \phi(b) \leq (1+\epsilon)v \) if \( \phi(b) > 0 \) and \( v = 0 \) otherwise. The time complexity of Algorithm 2 is \( O(\frac{1}{2} \log(\frac{B}{\epsilon})) \log B \cdot \theta \) if \( \phi \) has a positive value, where \( \delta \) and \( \Delta \) are the minimum and maximum positive values of \( \phi \), respectively, and \( \theta \) is the running time of evaluating \( \phi \), and is \( O(\log B \cdot \theta) \) otherwise.

Proof. Let \( S = \{(b_\tau, \tau)\} \) be the set of pairs output by Algorithm 2. Our reconstruction algorithm is as follows: Given \( b \in \{0,1,\ldots,B-1\} \), let \( (b_\tau, \tau^*) \) be the pair with the minimum \( b_\tau \), where \( b < b_\tau \). Note that such a \( b_\tau \) always exists because a pair of the form \( (B, \tau) \) is always added to \( S \). We then output \( \tau^* \). The time complexity of this reconstruction algorithm is clearly \( O(\log B) \).

We now show the correctness of the reconstruction algorithm. If \( \phi(b) > 0 \), then, in particular, we have \( \phi(b) \geq \delta \). Then, \( \tau^* \) is the maximum value of the form \( \delta(1+\epsilon)^k \) at most
Algorithm 2 Sketching subroutine for Algorithm 3

Input: $\phi : \{0, 1, \ldots, B - 1\} \rightarrow \mathbb{R}, \epsilon > 0$.

Assumption: $\phi$ is non-increasing.

1: $S \leftarrow \emptyset$. We regard $\phi(B) = -\infty$.
2: Find the minimum $b_0 \in \{0, 1, \ldots, B\}$ with $\phi(b_0) \leq 0$ by binary search.
3: if $b_0 \geq 1$ then # $\phi$ has a positive value.
   4: $\Delta \leftarrow \phi(0)$ and $\delta \leftarrow \phi(b_0 - 1)$.
   5: for $(\tau \leftarrow \delta; \tau \leq \Delta; \tau \leftarrow (1 + \epsilon) \tau)$ do
      6: Find the minimum $b_\tau \in \{0, 1, \ldots, B\}$ with $\phi(b_\tau) < \tau$ by binary search.
      7: $S \leftarrow S \cup \{(b_\tau, \tau)\}$
   8: end for
   9: if $b_\delta \neq B$ then
      10: $S \leftarrow S \cup \{(B, 0)\}$.
   11: end if
12: else # $\phi$ is non-positive.
   13: $S \leftarrow S \cup \{(B, 0)\}$.
14: end if
15: return $S$.

$\phi(b)$. Hence, we have $\tau^* \leq \phi(b) < (1 + \epsilon) \tau^*$. If $\phi(b) \leq 0$, $(b_\tau, \tau^*) = (B, 0)$ and we output zero.

Finally, we analyze the time complexity of Algorithm 2. Each binary search requires $O(\log B)$ time. The number of binary searches performed is $O(\log_{1 + \frac{\epsilon}{2}} \frac{1}{\delta}) = O\left(\frac{1}{\delta} \log \frac{1}{\delta}\right)$ when $\phi$ has a positive value and 1 if $\phi$ is non-positive. Hence, we have the desired time complexity.

We can regard Algorithm 2 as providing a value oracle for a function $\tilde{\phi} : \{0, 1, \ldots, B - 1\} \rightarrow \mathbb{R}^+$ that is an approximation to the input function $\phi : \{0, 1, \ldots, B - 1\} \rightarrow \mathbb{R}$.

We now describe our algorithm for maximizing DR-submodular functions. The basic idea is similar to Algorithm 1, but when we need $f(x|\chi_x)$ and $f(-\chi_x|y)$, we use approximations to them instead. Let $\alpha$ and $\beta$ be approximations to $f(x|\chi_x)$ and $f(-\chi_x|y)$, respectively, obtained by Algorithm 2. Then, we increase $x(e)$ by one with probability $\frac{\alpha}{\alpha + \beta}$ and decrease $y(e)$ by one with the complement probability $\frac{\beta}{\alpha + \beta}$. The details are given in Algorithm 3.

We now analyze Algorithm 3. An iteration refers to an iteration in the while loop from Line 5. We have $\|B\|_1$ iterations in total. For $k \in \{1, \ldots, \|B\|_1\}$, let $x_k$ and $y_k$ be $x$ and $y$, respectively, right after the $k$th iteration. Note that $x_k \|B\|_1 = y_k \|B\|_1$ is the output of the algorithm. We define $x_0 = 0$ and $y_0 = B$ for convenience.

Let $o$ be an optimal solution. For $k \in \{0, 1, \ldots, \|B\|_1\}$, we then define $o_k = o \cup x_k \cap y_k$, and $o_{\|B\|_1} = o \cup y_k \cap y_k$, respectively. Let $o_{\|B\|_1} = o \cup y_k \cap y_k$, equals the output of the algorithm. We have the following key lemma.

Lemma 3. For every $k \in \|B\|_1$, we have

$$E[f(o_k) - f(o_k)] = \frac{1}{2} \epsilon E[f(x_k) - f(x_{k-1}) + f(y_k) - f(y_{k-1})]$$

Proof. Fix $k \in \|B\|_1$ and let $e$ be the element of interest in the $k$th iteration. Let $\alpha$ and $\beta$ be the values in Line 6 in

Algorithm 3 Algorithm with polynomially many queries

Input: $f : \mathbb{Z}_+^E \rightarrow \mathbb{R}$, $B \in \mathbb{Z}_+^E$, $\epsilon > 0$.

Assumption: $f$ is DR-submodular.

1: $x \leftarrow 0$, $y \leftarrow B$.
2: for $e \in E$ do
   3: Define $g, h : \{0, 1, \ldots, B - 1\} \rightarrow \mathbb{R}$ as $g(x) = f(x|\chi_x)$ and $h(y) = f(-\chi_y|y)$.
   4: Let $\tilde{g}$ and $\tilde{h}$ be approximations to $g$ and $h$, respectively, given by Algorithm 2.
   5: while $x(e) < y(e)$ do
      6: $\alpha \leftarrow \tilde{g}(x(e))$ and $\beta \leftarrow \tilde{h}(B(e) - y(e))$.
      7: $x(e) \leftarrow x(e) + 1$ with probability $\frac{\alpha}{\alpha + \beta}$ and $y(e) \leftarrow y(e) - 1$ with the complement probability $\frac{\beta}{\alpha + \beta}$. If $\alpha = \beta = 0$, we assume $\frac{\alpha}{\alpha + \beta} = 1$.
   8: end while
   9: end for
10: return $x$.

the $k$th iteration. We then have

$$E[f(x_k) - f(x_{k-1}) + f(y_k) - f(y_{k-1})] = \frac{\alpha}{\alpha + \beta} f(x_k - x_{k-1}) + \frac{\beta}{\alpha + \beta} f(-x_{k-1}) - y_{k-1})$$

$$\geq \frac{\alpha}{\alpha + \beta} \leq \frac{\beta}{\alpha + \beta} = \frac{\alpha^2 + \beta^2}{\alpha + \beta},$$

where we use the guarantee in Lemma 2 in the inequality.

We next establish an upper bound of $E[f(o_k) - f(o_k)]$. As $o_k = (o \cup x_k) \cap y_k$, conditioned on a fixed $o_k$, we obtain

$$E[f(o_k) - f(o_k)] = \frac{\alpha}{\alpha + \beta} f(o_k - f(o_k))$$

$$\leq \frac{\beta}{\alpha + \beta} f(o_k) - f(o_k) \cap x_k(e) \chi_x)$$

$$+ \frac{\beta}{\alpha + \beta} f(o_k - f(o_k) \cap y_k(e) \chi_y).$$

Claim 4. (4) $\leq \frac{(1 + \epsilon) \alpha \beta}{\alpha + \beta}$.

Proof. We show this claim by considering the following three cases.

If $x_k(e) \leq o_k - 1 \leq y_k(e)$, then (4) is zero.

If $o_k - 1 < x_k(e)$, then $o_k(e) = o_k - 1(e) + 1$, and the first term of (4) is

$$f(o_k) - f(o_k) - f(x_k(e) \chi_x)$$

$$= f(o_k - 1 - f(o_k - 1(e) + \chi_x)$$

$$\leq f(y_k - 1 - f(y_k - 1)$$

$$= f(-x_k(e) - y_k(e)$$

$$\leq (1 + \epsilon) \beta.$$
We now return to proving Lemma 3. By Claim 4,
\[
(4) \leq \frac{(1 + \epsilon)\alpha \beta}{\alpha + \beta} \leq \frac{1 + \epsilon}{2} \left( \frac{\alpha^2 + \beta^2}{\alpha + \beta} \right) \leq \frac{1 + \epsilon}{2},
\]
which indicates the desired result. \( \square \)

**Theorem 5.** Algorithm 3 is a \( \frac{1}{2+\epsilon} \)-approximation algorithm for (1) with time complexity \( O(\|B\| \log\|B\|_\infty) \). The above sum is telescopic, and hence we obtain
\[
\sum_{k=1}^{\|B\|} E[f(o_{k-1}) - f(o_k)] 
\leq \frac{1 + \epsilon}{2} \sum_{k=1}^{\|B\|} E[f(x_k) - f(x_{k-1}) + f(y_k) - f(y_{k-1})].
\]

The polynomial-time algorithm

In many applications, the running time needed to evaluate the input function is a bottleneck, and hence Algorithm 3 is already satisfactory. However, it is theoretically interesting to reduce the total running time to a polynomial, and we show the following. The proof is deferred to the full version.

**Theorem 7.** There exists a \( \frac{1}{2+\epsilon} \)-approximation algorithm with time complexity \( O(\|B\| \log\|B\|_\infty \log\|B\|_\infty) \), where \( \delta \) and \( \Delta \) are the minimum positive marginal gain and the maximum positive value, respectively, of \( f \) and \( \theta \) is the running time of evaluating \( f \).

**Experiments**

In this section, we show our experimental results and the superiority of our algorithm with respect to other baseline algorithms.

**Experimental setting**

We conducted experiments on a Linux server with an Intel Xeon E5-2690 (2.90 GHz) processor and 256 GB of main memory. All the algorithms were implemented in C# and were run using Mono 4.2.3.

We compared the following four algorithms:

- **Single Greedy (SG):** We start with \( x = 0 \). For each element \( e \in E \), as long as the marginal gain of adding \( x_e \) to the current solution \( x \) is positive, we add it to \( x \). The reason that we do not choose the element with the maximum marginal gain is to reduce the number of oracle calls, and our preliminary experiments showed that such a tweak does not improve the solution quality.

- **Double Greedy (DG, Algorithm 1).**

- **Lattice Double Greedy (Lattice-DG):** The 1/3-approximation algorithm for maximizing non-monotone lattice submodular functions (Gottschalk and Peis 2015).

- **Double Greedy with a polynomial number of oracle calls with error parameter \( \epsilon > 0 \) (Fast-DG, Algorithm 3).**

We measure the efficiency of an algorithm by the number of oracle calls instead of the total running time. Indeed, the running time for evaluating the input function is the dominant factor of the total running time, because objective functions in typical machine learning tasks contain sums over all data points, which is time consuming. Therefore, we do not consider the polynomial-time algorithm (Theorem 7) here.

**Revenue maximization**

In this application, we consider revenue maximization on an (undirected) social network \( G = (V, W) \), where \( W = (w_{ij})_{i,j \in V} \) represents the weights of edges. The goal is to offer for free or advertise a product to users so that the revenue increases through their word-of-mouth effect on others. If we invest \( x \) units of cost on a user \( i \in V \), the user becomes an advocate of the product (independently from other users) with probability \( 1 - (1 - p)^x \), where \( p \in (0, 1) \) is a parameter. This means that, for investing a unit cost to \( i \), we have an extra chance that the user \( i \) becomes an advocate.
with probability $p$. Let $S \subseteq V$ be a set of users who advocate the product. Note that $S$ is a random set. Following a simplified version of the model introduced by (Hartline, Mirrokni, and Sundararajan 2008), the revenue is defined as $\sum_{i \in S} \sum_{j \in V \setminus S} w_{ij}$. Let $f : \mathbb{Z}_+ \to \mathbb{R}$ be the expected revenue obtained in this model, that is,
\[
    f(x) = \mathbb{E} \left[ \sum_{i \in S} \sum_{j \in V \setminus S} w_{ij} \right] = \sum_{i \in S} \sum_{j \in V \setminus S} w_{ij}(1 - (1 - p)^x(i))(1 - p)^x(j).
\]

It is not hard to show that $f$ is non-monotone DR-submodular function (see the full version for the proof).

In our experiment, we used three networks, Adolescent health (2,539 vertices and 12,969 edges), Advogato (6,541 vertices and 61,127 edges), and Twitter lists (23,370 vertices and 33,101 edges), all taken from (Kunegis 2013). We regard all the networks as undirected. We set $p = 0.0001$, and set $w_{ij} = 1$ when an edge exists between $i$ and $j$ and $w_{ij} = 0$ otherwise. We imposed the constraint $0 \leq x(e) \leq B$ for every $e \in E$, where $B$ is chosen from $\{10^2, \ldots, 10^6\}$.

Table 1 shows the objective values obtained by each method. As can be seen, except for Lattice-DG, which is clearly the worst, the choice of a method does not much affect the obtained objective value for all the networks. Notably, even when $\epsilon$ is as large as 0.5, the objective values obtained by Fast-DG are almost the same as SG and DG.

Figure 1 illustrates the number of oracle calls of each method. The number of oracle calls of DG and Lattice-DG is linear in $B$, whereas that of Fast-DG slowly grows. Although the number of oracle calls of SG also slowly grows, it is always orders of magnitude larger than that of Fast-DG with $\epsilon = 0.5$ or $\epsilon = 0.05$.

In summary, Fast-DG, achieves almost the best objective value, whereas the number of oracle calls is two or three orders of magnitude smaller than those of the other methods when $B$ is large.

Conclusions

In this paper, we proposed a polynomial-time $\frac{1}{2+\epsilon}$-approximation algorithm for non-monotone DR-submodular function maximization. Our experimental results on the revenue maximization problem showed the superiority of our method against other baseline algorithms.

Maximizing a submodular set function under constraints is well studied (Lee et al. 2009; Gupta et al. 2010; Chekuri, Vondrák, and Zenklusen 2014; Mirzasoleiman et al. 2016). An intriguing open question is whether we can obtain polynomial-time algorithms for maximizing DR-submodular functions under constraints such as cardinality constraints, polymatroid constraints, and knapsack constraints.

Acknowledgments T. S. is supported by JSPS Grant-in-Aid for Research Activity Start-up. Y. Y. is supported by JSPS Grant-in-Aid for Young Scientists (B) (No. 26730009), MEXT Grant-in-Aid for Scientific Research on Innovative Areas (No. 24106003), and JST, ERATO, Kawarabayashi Large Graph Project.

![Figure 1: Number of oracle calls](image-url)

<table>
<thead>
<tr>
<th>Method</th>
<th>Objective Value</th>
<th>Adolescent health</th>
<th>Advogato</th>
<th>Twitter lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG</td>
<td>280.55, 7331.42</td>
<td>2015.96, 7331.42</td>
<td>25516.05, 7331.42</td>
<td>25484.02</td>
</tr>
<tr>
<td>DG</td>
<td>280.55, 7331.42</td>
<td>2015.96, 7331.42</td>
<td>25516.05, 7331.42</td>
<td>25484.02</td>
</tr>
<tr>
<td>Fast-DG</td>
<td>280.55, 7331.42</td>
<td>2015.96, 7331.42</td>
<td>25516.05, 7331.42</td>
<td>25484.02</td>
</tr>
<tr>
<td>Fast-DG,0.5</td>
<td>280.55, 7331.42</td>
<td>2015.96, 7331.42</td>
<td>25516.05, 7331.42</td>
<td>25484.02</td>
</tr>
<tr>
<td>Fast-DG,0.05</td>
<td>280.55, 7331.42</td>
<td>2015.96, 7331.42</td>
<td>25516.05, 7331.42</td>
<td>25484.02</td>
</tr>
</tbody>
</table>
References


