MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

Faster Approximation Algorithms for Maximizing a Monotone Submodular Function Subject to a b-Matching Constraint

Kaito FUJII

(Communicated by Satoru IWATA)

METR 2015–12 March 2015

Faster Approximation Algorithms for Maximizing a Monotone Submodular Function Subject to a b-Matching Constraint

Kaito FUJII

Department of Mathematical Engineering and Information Physics
School of Engineering
The University of Tokyo
fujii.kaito@gmail.com

March 30, 2015

Abstract

Maximizing a monotone submodular function subject to a b-matching constraint is increasing in importance due to the application for the content spread maximization problem, but few practical algorithms are known other than the greedy algorithm. The best approximation scheme so far is the local search algorithm, proposed by Feldman, Naor, Schwartz, Ward (2011). It obtains a $1/(2+\frac{1}{k}+\epsilon)$-approximation for an arbitrary positive integer k and positive real number ϵ. For graphs with n vertices and m edges, the running time of the local search algorithm is $\mathrm{O}(b^{k+1}n^{k+1}m\epsilon^{-1})$, which is impractical for large problems.

In this paper, we present two new algorithms for this problem. One is the find walk algorithm that runs in $\mathrm{O}(bm)$ time and achieves a 1/4-approximation. It is faster than the greedy algorithm whose approximation ratio is 1/3. The other is the randomized local search algorithm that is a faster variant of the local search algorithm in the case of $k = 2$. In expectation, it runs in $\mathrm{O}(b^3 mn \log \frac{1}{\epsilon})$ time and obtains a $(2/5 - \epsilon)$-approximate solution.

1 Introduction

Maximizing a monotone submodular function has been attracting much attention recently. This is motivated by various applications in many fields. For example, the social welfare maximization problem, which is an important problem in combinatorial auction, is a special case of maximization under a matroid constraint [14]. Another application involves social networks. The influence of a set of nodes can be described as a monotone submodular function [12]. There are other applications such as sensor placements [10] and document summarization [15].

The content spread maximization problem is one such applications. It is introduced by Chaoji, Ranu, Rastogi, Bhatt [3] for friend suggestions in social networks. It considers which pairs should come to be friends in order to maximize the spreadability of contents. This problem can be reduced to maximizing a submodular function subject to a b-matching constraint, which we deal with in this paper.

There are several approaches to this problem. The most simple approach is to use the classical greedy algorithm that was introduced in Fisher, Nemhauser, Wolsey [8] for general constraints. Jenkyns [11] suggested that this algorithm achieves a $1/(p+1)$-approximation for a p-system constraint, and Calinescu, Chekuri, Pál, Vondrák [2] gave the full proof. A b-matching constraint is a special case of a 2-system constraint, so that the greedy algorithm can be used for b-matching constraints. This algorithm can achieve a 1/3-approximation in $\mathrm{O}(bnm)$ time, where n is the number of vertices and m is the number of edges.

The greedy algorithm with decreasing threshold was devised by Badanidiyuru, Vondrák [1]. This algorithm speeds up the greedy algorithm while almost keeping the approximation ratio. Specifically, it can obtain a $(1/3-\epsilon)$-approximation in $\mathrm{O}(\frac{m}{\epsilon}\log\frac{m}{\epsilon})$ time.

Feldman, Naor, Schwartz, Ward [7] designed the local search algorithm, which is a $1/(p+\frac{1}{k}+\epsilon)$-approximation for a p-exchange system constraint, where k is an arbitrary positive integer and ϵ is an arbitrary positive real number. A b-matching constraint is a special case of a 2-exchange system, therefore a $1/(2+\frac{1}{k}+\epsilon)$-approximation can be achieved. In the case of $k=1$, the approximation ratio is almost equal to the greedy algorithm, and as k becomes larger, the approximation ratio increases asymptotically toward 1/2, which is the best known so far. However, the running time is $\mathrm{O}(b^{k+1}n^{k+1}m\epsilon^{-1})$ and not practical for large graphs.

Another approach is the continuous greedy algorithm and pipage rounding. In this approach, we obtain an approximate solution of a continuous relaxation problem by the continuous greedy algorithm and round it to get an integer solution. It is first introduced by [2] as a $(1-1/\mathrm{e})$-approximation algorithm for a matroid constraint. The continuous greedy algorithm can be applied to various constraints including b-matching constraints, but there is no efficient rounding method for a b-matching constraint. [3] proposed a simple rounding method for a b-matching constraint, but the overall running time is $\tilde{\mathrm{O}}(n^7)$ and the approximation ratio is worse than that of the greedy algorithm.

We present two new approximation algorithms for maximizing a monotone submodular function subject to a b-matching constraint.

One is the find walk algorithm. This approach was first proposed by

Drake, Hougardy [6] in the design of a linear time 1/2-approximation algorithm for the maximum weighted matching problem. Mestre [16] extended it to the b-matching problem for any positive integer b while keeping the approximation ratio and the time complexity. We show that this approach can be used to derive a 1/4-approximation scheme for a monotone submodular function.

The other is a randomized local search algorithm. This approach was first used by Pettie, Sanders [18] for the maximum weighted matching problem and extended by Mestre [16] for the b-matching. For both problems, the resulting algorithms run in $\mathrm{O}(bm \log \frac{1}{\epsilon})$ time in expectation and return a $(2/3 - \epsilon)$-approximation in expectation. We extend this approach for maximizing a monotone submodular function and prove that the approximation ratio is $(2/5 - \epsilon)$ and the running time is $\mathrm{O}(b^3 nm \log \frac{1}{\epsilon})$ in expectation.

2 Preliminaries

2.1 Monotone Submodular Functions

Let E be a finite set. First of all, we define some notation. For an arbitrary subset $S \subseteq E$ and an arbitrary element $e \in E$, let $S + e = S \cup \{e\}$ and $S - e = S \setminus \{e\}$. Similarly, $f(e) = f(\{e\})$ for any set function $f : 2^E \to \mathbb{R}_{\geq 0}$ and $e \in E$. Given a set function $f : 2^E \to \mathbb{R}_{\geq 0}$ and a subset $A \subseteq E$, $f_A : 2^E \to \mathbb{R}_{\geq 0}$ is defined by $f_A(S) = f(A \cup S) - f(A)$ for every set $S \subseteq E$.

A set function $f : 2^E \to \mathbb{R}_{\geq 0}$ is *submodular* if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq E$. For each $A \subseteq E$, f_A is also submodular. A set function $f : 2^E \to \mathbb{R}_{\geq 0}$ is *monotone* if $f(A) \leq f(B)$ for all $A \subseteq B \subseteq E$. It is widely known that f is submodular if and only if $f_A(e) \geq f_B(e)$ for all $A \subseteq B \subseteq E$ and $e \in E$.

We assume that f is normalized, i.e., $f(\emptyset) = 0$. Given a monotone submodular function f that is not normalized, we can obtain a normalized monotone submodular function $\tilde{f}$ by letting $\tilde{f}(S) = f(S) - f(\emptyset)$ for all $S \subseteq E$.

We use the following two properties of submodular functions proved by Lee, Sviridenko, Vondrák [13].

Lemma 2.1. ([13, Lemma 1.1])

Let f be a non-negative submodular function on N. Let $S' \subseteq S \subseteq N$, and let $\{T_l\}_{l=1}^t$ be a collection of subsets of $S \setminus S'$ such that each element of $S \setminus S'$ appears in exactly k of these subsets. Then

$$\sum_{l=1}^{t} [f(S) - f(S \setminus T_l)] \leq k \left(f(S) - f(S') \right).$$

Lemma 2.2. ([13, Lemma 1.2])

Let f be a non-negative submodular function on N. Let $S \subseteq N$ and $C \subseteq N$, and let $\{T_l\}_{l=1}^{t}$ be a collection of subsets of $C \setminus S$ such that each element of $C \setminus S$ appears in exactly k of these subsets. Then

$$\sum_{l=1}^{t} [f(S \cup T_l) - f(S)] \geq k\,(f(S \cup C) - f(S)) .$$

2.2 b-matching Constraints

Let $G = (V, E)$ be an undirected graph and b a positive integer. A subgraph of G is a *b-matching* if the degree of each vertex is no more than b. Under a b-matching constraint, a solution set $S \subseteq E$ must satisfy the condition that (V, S) is a b-matching of the graph. We define $\mathcal{I}$ as a set family of all sets of edges satisfying a b-matching constraint.

Let n be the number of vertices and m the number of edges. Let $\delta_E(v)$ denote the degree of vertex $v \in V$ in graph (V, E). Let E_v be the set of all edges incident to v in E for each vertex $v \in V$.

From now on, we deal with the monotone submodular function maximization subject to a b-matching constraint. In this problem, the ground set E of the objective function f is the set of all edges in the graph that represents the constraint, and if we regard f as a weight function on edges, the solution is a set of edges that has the maximum weight. Assume that the structure of the graph that represents the constraint is known a priori and can be used in algorithms.

When considering the time complexity of algorithms, we assume the value oracle and the independence oracle. In this assumption, we can obtain the value $f(S)$ and judge whether $S \in \mathcal{I}$ or not in $\mathrm{O}(1)$ time for an arbitrary set $S \subseteq E$.

3 The Find Walk Algorithm

The find walk algorithm was first introduced by Drake and Hougardy [6] as a linear time 1/2-approximation scheme for the maximum weighted matching problem. Mestre [16] proved that it can be generalized for the maximum weighted b-matching problem with the same approximation ratio. Here, we extend this algorithm to monotone submodular functions and prove that it returns a 1/4-approximate solution.

This algorithm runs as follows. To begin with, initialize some variables. Let $S := \emptyset$ and $c(u) := b$ for all vertices $u \in V$. Then, find a walk from an arbitrary vertex that satisfies $c(u) > 0$ and $\delta_E(u) > 0$, using $\mathsf{FindWalk}$ procedure iteratively.

In $\mathsf{FindWalk}(u)$ procedure, start with reducing $c(u)$ by one. Next, find the edge (u, v) that has the maximum marginal value $f_S((u, v))$ in all edges incident to u. If there remains no edge incident to u, make u be the end vertex of the current walk. After finding the edge of the maximum marginal value, remove it from E, and move the focus to u. If $c(u) = 0$, delete all edges incident to u from E. Then, repeat FindWalk procedure until we come to a dead end.

If one walk ends, find another walk from a vertex such that $c(u) > 0$ and $\delta_E(u) > 0$, and repeat. When there is no vertex such that $c(u) > 0$ and $\delta_E(u) > 0$, partition S into S_1 and S_2 so that an edge added to S in odd numbered time is taken into S_1, and even numbered time into S_2. Since the number of calling FindWalk(u) is no more than b for all $u \in V$, and at each calling, the degree of the vertex increases by at most two, it follows that both S_1 and S_2 are b-matchings. Then, return $\text{argmax}\{f(S_1), f(S_2)\}$ as an output.

Algorithm 1 Find Walk Algorithm

```
1: $S := \emptyset$.
2: $c(u) := b$ for all $u \in V$.
3: **while** there is $u \in V$ such that $c(u) > 0$ and $\delta_E(u) > 0$ **do**
4:     $S := S \cup \mathsf{FindWalk}(u)$.
5: split $S$ into $S_1$ and $S_2$.
6: **return** $\text{argmax}\{f(S_1), f(S_2)\}$
```

Algorithm 2 FindWalk(u)

```
1: $c(u) := c(u) - 1$.
2: **if** $\delta_S(u) = 0$ **then**
3:     **return** $\emptyset$.
4: $(u, v) := \text{argmax}\{f_S(e) \mid e \in E_u\}$.
5: $E := E - (u, v)$.
6: **if** $c(u) = 0$ **then**
7:     $E := E \setminus E_u$.
8: **return** $(u, v) + \mathsf{FindWalk}(v)$.
```

Theorem 3.1. The find walk algorithm runs in $\mathrm{O}(bm)$ time and returns a 1/4-approximate solution.

Proof. At the end of the algorithm, $\delta_E(u) = 0$ or $c(u) = 0$ for all vertices $u \in V$. For each edge (u, v), if $c(u) = 0$ or $c(v) = 0$, (u, v) was removed from E when $c(u)$ or $c(v)$ became 0. If $\delta_E(u) = 0$ and $\delta_E(v) = 0$, (u, v) must have been removed from E. Then, E is empty at the end of the algorithm.

Let S^* be an optimal set. For the proof, assume that S^* is known from the beginning of the algorithm and we consider making a map $\phi : S^* \to S$ during the algorithm as follows. When we add (u, v) to S, if $(u, v) \in S^*$ then let $\phi((u, v)) = (u, v)$, else choose an arbitrary edge $(u, \tilde{v}) \in S^*$ and let

$\phi((u,\tilde{v})) = (u,v)$. Then, remove the assigned edge from S^*. If $\delta_{S^*}(u) = 0$, assign no edge.

First, we show that when the algorithm stops, the map ϕ is completed, i.e., $S^* = \emptyset$. For all $(u,v) \in S^*$, if $c(u) = 0$ then u is passed b times and all edges in S^* incident to u are assigned. The same applies to the case $c(v) = 0$. At the end of the algorithm, $c(u) = 0$ or $c(v) = 0$ holds. This is because if $c(u) \neq 0$ and $c(v) \neq 0$, (u,v) is not removed and we can make a new walk from u or v.

Next, we show that $f_S(S^*) \leq f(S)$. For all $e \in S$, let S_e be S just before e is added to S, then we obtain:

$$f_S(S^*) \leq \sum_{e \in S^*} f_S(e) \leq \sum_{e \in S^*} f_{S_{\phi(e)}}(e) \leq \sum_{e \in S^*} f_{S_{\phi(e)}}(\phi(e)) \leq \sum_{e \in S} f_{S_e}(e) = f(S).$$

The first and second inequalities are due to the submodularity. The last inequality holds because we chose the edge that has the maximum marginal value at each step.

Due to the monotonicity, $f(S^*) - f(S) \leq f(S^* \cup S) - f(S) \leq f(S)$, and hence $f(S^*) \leq 2f(S)$. From the submodularity, $f(S) \leq f(S_1) + f(S_2)$, and it follows that $\max\{f(S_1), f(S_2)\} \geq \frac{1}{4} f(S^*)$.

The running time is bounded as follows. The number of calling $\mathsf{FindWalk}(u)$ is no more than b for each vertex $u \in V$, and the marginal value of each edge (u,v) is evaluated only in $\mathsf{FindWalk}(u)$ and $\mathsf{FindWalk}(v)$, so that the number of oracle calls for each edge is $\mathrm{O}(b)$. Therefore, the overall running time is $\mathrm{O}(bm)$. □

4 The Randomized Local Search

Feldman, Naor, Schwartz, Ward [7] designed the local search algorithm for maximizing various functions under a p-exchange system constraint. For a monotone submodular function, the approximation ratio of this algorithm is $1/(p + \frac{1}{k} + \epsilon)$ for an positive integer parameter k. Since b-matching constraints are a subset of 2-exchange systems, we can obtain a $1/(2 + \frac{1}{k} + \epsilon)$-approximation by using the local search algorithm. This is almost the same as the greedy algorithm when $k = 1$, and better when $k > 2$. But the local search algorithm is slower and not very practical.

We propose a randomized approximation scheme for this problem, which can get a $(2/5 - \epsilon)$-approximation in expectation. This approximation ratio is the same as that of the local search in the case of $k = 2$. The running time of the new algorithm is $\mathrm{O}(b^3 mn \log \frac{1}{\epsilon})$ in expectation. On the other hand, the local search algorithm in the case of $k = 2$ runs in $\mathrm{O}(b^3 n^3 m \epsilon^{-1})$ time even if we consider that we should check only alternating paths at each step of improvement[1]. Thus, our new algorithm is faster than the local search in

[1]In the local search algorithm, we can obtain the same approximation ratio if restrict

the case of $k = 2$.

The randomized local search was first devised by Pettie, Sanders [18] as a $(2/3 - \epsilon)$-approximation for maximizing a linear function subject to the maximum weighted matching problem. Mestre [16] extended it to b-matching constraints while keeping the approximation ratio. We show that this approach can be applied to monotone submodular functions and the approximation ratio is $2/5 - \epsilon$. This algorithm also can be interpreted as an accelerated variant of the local search algorithm in the case of $k = 2$ by randomizing.

Before explaining the algorithm, we define some notation. The randomized local search considers alternating paths, as with the local search algorithm, and for an arbitrary alternating path A, we call $f(S \triangle A) - f(S)$ the benefit of A. At each step of improvement, we consider two kinds of alternating paths: pieces and arms introduced by Mestre [16]. An arm A out of an vertex u is an alternating path that consists of an edge $(u, v) \in E \setminus S$ and maybe an edge $(v, x) \in S$. A piece P about an edge $(u, v) \in S$ is an alternating path that consists of (u, v) itself, maybe an arm out of u and maybe an arm out of v. We assume that an alternating cycle including e whose length is four is also a piece about e. In the algorithm and the proof, we consider only feasible exchanges, in other words, an arm or a piece A such that $S \triangle A \in \mathcal{I}$.

The randomized local search starts by obtaining an initial feasible set using the greedy algorithm and substituting it for the current solution S. It then repeats an improvement step $(\frac{bn}{5} \log \frac{1}{\epsilon})$ times. At each step, we pick up one vertex u out of n vertices uniformly at random, and choose case 1 with probability $\delta_S(u)/b$ or case 2 otherwise. If case 1 is chosen, pick an edge e out of all edges incident to u in S uniformly at random and find the most beneficial piece about e. If case 2 is chosen, find the most beneficial arm out of u. Then improve the current solution with the selected piece or arm.

Algorithm 3 Randomized Local Search

```
1: $S := \emptyset$.
2: for ($i := 0$; $i < \frac{bn}{5} \log \frac{1}{\epsilon}$; $i := i + 1$) do
3:     choose a vertex $u$ out of $V$ uniformly at random.
4:     if with probability $\delta_S(u)/b$ then
5:         choose an edge $e \in S$ out of edges incident to $u$ uniformly at random.
6:         find the most beneficial piece about $e$ and improve $S$ with it.
7:     else
8:         find the most beneficial arm out of $u$ and improve $S$ with it.
9: return $S$.
```

the searching area to alternating paths. Feldman, Naor, Schwartz, Ward [7] did not claim this fact explicitly, but it follows naturally from their proof.

Theorem 4.1. The randomized local search algorithm runs in $\mathrm{O}(b^3 mn \log \frac{1}{\epsilon})$ time in expectation and returns a $(2/5 - \epsilon)$-approximate solution in expectation.

Proof. In this proof, let S^* be one of the optimal sets, and S the current solution. Both S^* and S are feasible.

First, we evaluate the approximation ratio of this algorithm. The expected value of one step benefit can be bounded from below by the average benefit of a certain multiset of alternating paths.

We show that $S \triangle S^*$ can be partitioned into alternating paths satisfying these conditions:

- for any vertex u such that $\delta_S(u) > \delta_{S^*}(u)$, there are exactly $\delta_S(u) - \delta_{S^*}(u)$ alternating paths that starts with or ends at a edge $(u, v) \in S \setminus S^*$, and
- for any vertex u such that $\delta_{S^*}(u) > \delta_S(u)$, there are exactly $\delta_{S^*}(u) - \delta_S(u)$ alternating paths that starts with or ends at a edge $(u, v) \in S^* \setminus S$.

We start with non-cycle paths. If there is a vertex u such that $\delta_S(u) > \delta_{S^*}(u)$, start a path with an arbitrary edge $(u, v) \in S \setminus S^*$. Then, we move the focus to v. If $\delta_S(v) \leq \delta_{S^*}(v)$, there should be an edge $(v, x) \in S^* \setminus S$, and we can move the focus to the next vertex x. If $\delta_S(v) > \delta_{S^*}(v)$, end this path and let v be the other end vertex. Repeat this process exchanging $S \setminus S^*$ and $S^* \setminus S$ each time until the path ends. After that, remove this path from $S \triangle S^*$ and repeatedly get such paths until there is no vertex such that $\delta_S(u) > \delta_{S^*}(u)$. Then, obtain alternating paths from vertex u such that $\delta_{S^*}(u) > \delta_S(u)$ in the same way.

Next, we get alternating cycles. From an arbitrary vertex u such that $\delta_S(u) > 0$, we find an alternating path in the same way as before. For each vertex v, $\delta_S(v) = \delta_{S^*}(v)$ holds and it follows that this path does not terminate until it reaches the start vertex. Since the number of remaining edges decreases monotonically, $S \triangle S^*$ will be empty at the end. Therefore, $S \triangle S^*$ can be partitioned into alternating paths satisfying the above conditions. Let $A_1, \cdots, A_h$ be these paths.

We will next consider obtaining a multiset $\mathcal{P} = \{P_1, \cdots, P_l\}$ and a map $\phi : \mathcal{P} \to (S \setminus S^*) \cup V$ such that each P_i is an arm or a piece, and ϕ satisfies the following two conditions:

- $\forall e \in S \setminus S^*, \ |\phi^{-1}(e)| \leq 2,$
- $\forall u \in V, \ |\phi^{-1}(u)| \leq b - \delta_S(u).$

For each A_i, we consider separately whether A_i is a cycle or not. If A_i is not a cycle, for all $e \in (S \setminus S^*) \cap A_i$, take the piece P from e that is a subpath of A_i into $\mathcal{P}$ and let $\phi(P) = e$. In addition, if the end edge e of A_i

is an element of $S^* \setminus S$, take the arm A that consists of e and the next edge in A_i into $\mathcal{P}$ and assign $\phi(A)$ to the end vertex u. Next, we consider the case where A_i is a cycle. If $|A_i| = 4$, assume $\{e_1, e_2\} = A_i \cap (S \setminus S^*)$, then duplicate A_i as C_1, C_2, take them into $\mathcal{P}$, and let $\phi(C_1) = e_1$ and $\phi(C_2) = e_2$. If $|A_i| \geq 6$, for all $e \in A_i \cap (S \setminus S^*)$, let P be the subpath of A_i that is also a piece about e and add P to $\mathcal{P}$. Then let $\phi(P) = e$.

We show that ϕ fulfills the above two conditions. Since $A_1, \cdots, A_h$ is the partition of $S \triangle S^*$, each $e \in S \setminus S^*$ belongs to only one A_i. Hence ϕ^{-1} is a singleton of the piece about e. This is the reason the first condition is satisfied. On the other hand, $\phi^{-1}(u)$ consists of arms out of u in $\mathcal{P}$, and the number of such arms is the number of A_i that starts with or ends at a edge $(u, v) \in S^* \setminus S$ with a certain v. Hence, $|\phi^{-1}(u)|$ is $\max\{\delta_{S^*}(u) - \delta_S(u), 0\}$. From $\delta_{S^*}(u) \leq b$ and $\delta_S(u) \leq b$, the second condition follows.

Next, we consider the number of occurrences of each edge $(u, v) \in S \triangle S^*$ in $\mathcal{P}$. For any $(u, v) \in S^* \setminus S$, if there is no neighboring edge of (u, v) on the u side in A_i, there is an arm out of u including (u, v) in $\mathcal{P}$. If the next edge (u, x) exists, $\mathcal{P}$ has a piece about (u, x) including (u, v). The same is true with the v side. In addition, (u, v) does not appear in other paths of $\mathcal{P}$. Accordingly, (u, v) appears in exactly 2 alternating paths of $\mathcal{P}$. For any $(u, v) \in S \setminus S^*$, there is a piece about (u, v) in $\mathcal{P}$. If (u, v) has a neighboring edge (x, u) in A_i, there is an arm out of x or a piece about the edge next to (x, u) in $\mathcal{P}$. The same is true with the v side, and there is no other path including (u, v) in $\mathcal{P}$. Consequenetly, (u, v) appears in at most 3 alternating paths of $\mathcal{P}$.

Let P_e be the most beneficial arm about $e \in S$ and P_u be the most beneficial piece out of $u \in V$. Using the above decomposition, we can bound the expected value of improvement at each step:

$$\begin{aligned}
&\mathbb{E}[\text{one step benefit}] \\
&= \frac{1}{n} \sum_{u \in V} \left\{ \frac{1}{b} \sum_{e \in S_u} (f(S \triangle P_e) - f(S)) + \left(1 - \frac{\delta_S(u)}{b}\right) (f(S \triangle P_u) - f(S)) \right\} \\
&\geq \frac{1}{bn} \left\{ \sum_{e \in S} 2 \left(f(S \triangle P_e) - f(S)\right) + \sum_{u \in V} (b - \delta_S(u)) \left(f(S \triangle P_u) - f(S)\right) \right\} \\
&\geq \frac{1}{bn} \left\{ \sum_{e \in S \setminus S^*} \sum_{P \in \mathcal{P} : \phi(P) = e} (f(S \triangle P) - f(S)) + \sum_{u \in V} \sum_{P \in \mathcal{P} : \phi(P) = u} (f(S \triangle P) - f(S)) \right\} \\
&\geq \frac{1}{bn} \sum_{P \in \mathcal{P}} \{f(S \triangle P) - f(S)\} \\
&\geq \frac{1}{bn} \sum_{P \in \mathcal{P}} \{f(S \cup P)) - f(S) + f(S \setminus P) - f(S)\}.
\end{aligned}$$

The last inequality holds because of $f(S \triangle P) - f(S \setminus P) \geq f(S \cup P) - f(S)$, which is obtained by the submodularity.

Now we use the two lemmas 2.1 and 2.2. Each element of $S \setminus S^*$ appears in less than three elements of $\mathcal{P}$ and each element of $S^* \setminus S$ appears in exactly two elements of $\mathcal{P}$. Using the above two lemmas, we obtain:

$$\begin{aligned}
&\mathbb{E}[\text{one step benefit}] \\
&\geq \frac{1}{bn}\left\{2f_S(S^*) - 3f(S)\right\} \\
&\geq \frac{1}{bn}\left\{2f(S^*) - 5f(S)\right\} \\
&= \frac{5}{bn}\left\{\frac{2}{5}f(S^*) - f(S)\right\}.
\end{aligned}$$

The second inequality is due to the monotonicity.

If $f(S) \geq \frac{2}{5}f(S^*)$, the solution is already a 2/5-approximation. If $f(S) < \frac{2}{5}f(S^*)$, through one step of this algorithm, the difference between $f(S)$ and $\frac{2}{5}f(S^*)$ decreases at least a factor $(1 - \frac{5}{bn})$. At the beginning of the algorithm, $\frac{2}{5}f(S^*) - f(S)$ is no more than $\frac{2}{5}f(S^*)$, so that after $(\frac{bn}{5}\log\frac{1}{\epsilon})$ times of improvement, S satisfies:

$$\frac{2}{5}f(S^*) - f(S) \leq \frac{2}{5}f(S^*) \cdot \left(1 - \frac{5}{bn}\right)^{\frac{bn}{5}\log\frac{1}{\epsilon}} \leq \epsilon f(S^*).$$

We conclude that we obtain a $(\frac{2}{5} - \epsilon)$-approximation in expectation.

Next, we evaluate the running time. For an arbitrary $u \in V$ and an arbitrary edge e incident to u, the time of finding the most beneficial piece about e is $\mathrm{O}(\delta_E(u)b^2n)$ and the time of finding the most beneficial arm out of u is $\mathrm{O}(\delta_E(u)b)$. Using this, we can bound the expected value of the running time at each step:

$$\begin{aligned}
&\mathbb{E}[\text{one step time}] \\
&= \frac{1}{n}\sum_{u \in V}\left\{\frac{\delta_S(u)}{b}\delta_E(u)b^2n + \left(1 - \frac{\delta_S(u)}{b}\right)\delta_E(u)b\right\} \\
&\leq \frac{1}{n}\sum_{u \in V}\delta_E(u)b^2n \\
&\leq \frac{1}{n} \cdot 2mb^2n \\
&= \mathrm{O}(b^2m).
\end{aligned}$$

We need $(\frac{bn}{5}\log\frac{1}{\epsilon})$ times of updates, so that the running time is $\mathrm{O}(b^3mn\log\frac{1}{\epsilon})$ in expectation. □

5 Conclusion

In this paper, we proposed two approximation algorithms for maximizing monotone submodular functions subject to a b-matching constraint. One is the find walk algorithm that is executed in linear time and returns a 1/4-approximation, and the other is the randomized local search algorithm that is a faster variant of the local search algorithm with the parameter $k = 2$. It is an open problem whether the same result can be obtained using a similar randomization when $k > 2$.

Acknowledgement

This paper is based on the author's bachelor thesis at the University of Tokyo. The author thanks the thesis adviser, Satoru Iwata, very much for his helpful guidance. I would like to thank Jan Vondrák for informing us of the previous work [7]. I also thank Yuji Nakatsukasa and Tasuku Soma for the inspiring discussions.

References

[1] A. Badanidiyuru, and J. Vondrák, Fast algorithms for maximizing submodular functions, *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1497–1514, 2014.

[2] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, Maximizing a submodular set function subject to a matroid constraint, *SIAM Journal on Computing*, 40, pp. 1740–1766, 2011.

[3] V. Chaoji, S. Ranu, R. Rastogi, and R. Bhatt, Recommendations to boost content spread in social networks, *Proceedings of 21st International World Wide Web Conference*, pp. 529–538, 2012.

[4] C. Chekuri, J. Vondrák, and R. Zenklusen, Dependent randomized rounding via exchange properties of combinatorial structures, *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, pp. 575–584, 2010.

[5] C. Chekuri, J. Vondrák, and R. Zenklusen, Multi-budgeted matchings and matroid intersection via dependent rounding, *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithm*, pp. 1080–1097, 2010.

[6] D. E. Drake, and S. Hougardy, A simple approximation algorithm for the weighted matching problem, *Information Proceedings Letters*, 85(4), pp. 211–213, 2003.

[7] M. Feldman, J. Naor, R. Schwartz, and J. Ward, Improved approximations for k-exchange systems, *Proceedings of 19th Annual European Symposium on Algorithms*, pp. 784–798, 2011.

[8] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, An analysis of approximations for maximizing submodular set functions. II., *Mathematical Programming Studies*, no. 8, pp. 73–87, 1978.

[9] H. N. Gabow, An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pp. 448–456, 1983.

[10] Carlos Guestrin, Andreas Krause, and Ajit Paul Singh, Near-optimal sensor placements in gaussian processes, *Proceedings of the 22nd International Conference on Machine Learning*, pp. 265–272, 2005.

[11] T. Jenkyns, The efficacy of the greedy algorithm, *Proceedings of the 7th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pp. 341–350, 1976.

[12] D. Kempe, J. M. Kleinberg, and É. Tardos, Maximizing the spread of influence through a social network, *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.137–146, 2003.

[13] J. Lee, M. Sviridenko, and J. Vondrák, Submodular maximization over multiple matroids via generalized exchange properties, *Mathematics of Operations Research* 35, pp. 795–806, 2010.

[14] B. Lehmann, D. Lehmann, and N. Nisan, Combinatorial auctions with decreasing marginal utilities, *Games and Economic Behavior*, 55(2): pp.270–296, 2006.

[15] H. Lin, and J. Bilmes, Multi-document summarization via budgeted maximization of submodular functions, *Proceedings of Human Language Technologies: The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp.912–920, 2010.

[16] J. Mestre, Greedy in approximation algorithms, *Proceedings of 14th Annual European Symposium on Algorithms*, pp. 528–539, 2006.

[17] M. Minoux, Accelerated greedy algorithms for maximizing submodular set functions, *Optimization Techniques*, J. Stoer, ed., Springer-Verlag, Berlin, pp. 234–243, 1977.

[18] S. Pettie, and P. Sanders, A simpler linear time $2/3 - \epsilon$ approximation to maximum weighted matching, *Information Processing Letters*, 91(6): pp.271–276, 2004.