Generalized Assignment via Submodular Optimization with Reserved Capacity

Ariel Kulik
Computer Science Department, Technion, Haifa, Israel
kulik@cs.technion.ac.il

Kanthi Sarpatwar
IBM Research, Yorktown Heights, NY, USA
sarpatwa@us.ibm.com

Baruch Schieber
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA
sbar@njit.edu

Hadas Shachnai
Computer Science Department, Technion, Haifa, Israel
hadas@cs.technion.ac.il

Abstract

We study a variant of the *generalized assignment problem* (GAP) with group constraints. An instance of Group GAP is a set I of items, partitioned into L groups, and a set of m uniform (unit-sized) bins. Each item $i \in I$ has a size $s_i > 0$, and a profit $p_{i,j} \geq 0$ if packed in bin j. A group of items is *satisfied* if all of its items are packed. The goal is to find a feasible packing of a subset of the items in the bins such that the total profit from satisfied groups is maximized. We point to central applications of Group GAP in Video-on-Demand services, mobile Device-to-Device network caching and base station cooperation in 5G networks.

Our main result is a $\frac{1}{6}$-approximation algorithm for Group GAP instances where the total size of each group is at most $\frac{m}{2}$. At the heart of our algorithm lies an interesting derivation of a submodular function from the classic LP formulation of GAP, which facilitates the construction of a high profit solution utilizing at most half the total bin capacity, while the other half is *reserved* for later use. In particular, we give an algorithm for submodular maximization subject to a knapsack constraint, which finds a solution of profit at least $\frac{1}{3}$ of the optimum, using at most half the knapsack capacity, under mild restrictions on element sizes. Our novel approach of submodular optimization subject to a knapsack *with reserved capacity* constraint may find applications in solving other group assignment problems.

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Submodular optimization and polymatroids; Mathematics of computing → Linear programming; Mathematics of computing → Approximation algorithms

Keywords and phrases Group Generalized Assignment Problem, Submodular Maximization, Knapsack Constraints, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.69

Related Version A full version of the paper is available at `https://arxiv.org/abs/1907.01745`.

Acknowledgements H. Shachnai's work was conducted during a visit to DIMACS partially supported by the National Science Foundation under grant number CCF-1445755.

27th Annual European Symposium on Algorithms (ESA 2019).
Editors: Michael A. Bender, Ola Svensson, and Grzegorz Herman; Article No. 69; pp. 69:1–69:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

With the rapid adoption of cloud computing, wireless networks, and other modern platforms, resource allocation problems of various flavors have regained importance. One classic example is the *generalized assignment problem* (GAP). We are given a set of n items and m bins, $[m] = \{1, 2, \ldots, m\}$. Each item $i \in [n]$ has a size $s_{i,j} > 0$ and a profit $p_{i,j} \geq 0$ when packed into bin $j \in [m]$. The goal is to feasibly pack in the bins a subset of the items of maximum total profit. GAP has been widely studied, with applications ranging from grouping and loading in manufacturing systems to land use optimization in regional planning (see, e.g., [2, 10]). In discrete optimization, GAP has received considerable attention also as a special case of the *separable assignment* problem and *submodular maximization* (see, e.g., [23, 14, 4, 5]). We consider a variant of GAP with group constraints. An instance of Group GAP consists of a set $I = \{1, 2, \ldots, n\}$ of n items and m uniform (unit-sized) bins $M = \{1, \ldots, m\}$. Each item $i \in I$ has a size $s_i > 0$ and a profit $p_{i,j} \geq 0$ when assigned to bin $j \in [m]$. The items in I are partitioned into $L \geq 1$ groups, $\mathcal{G} = \{G_1, \ldots, G_L\}$. Given an assignment of items to bins, we say that a group is *satisfied* if all of its items are assigned. The goal is to find a feasible assignment of a subset of the items to bins such that the total profit from satisfied groups is maximized. Formally, a feasible assignment is a tuple $(U_1, \ldots, U_m)$, such that $U_j \cap U_k = \emptyset$ for all $1 \leq j < k \leq m$, $U_j \subseteq I$ and $\sum_{i \in U_j} s_i \leq 1$, for all $1 \leq j \leq m$. Let $I(U) = \cup_j U_j$. Then, $G_\ell \in \mathcal{G}$ is satisfied if $G_\ell \subseteq I(U)$. Let $\mathcal{G}_s = \{G_{\ell_1}, \ldots G_{\ell_t}\}$ be the set of satisfied groups and $I(\mathcal{G}_s) = \cup_{G_\ell \in \mathcal{G}_s} G_\ell$. Then we seek an assignment $(U_1, \ldots, U_m)$ for which $\sum_{j=1}^{m} \sum_{i \in U_j \cap I(\mathcal{G}_s)} p_{i,j}$ is maximized.

The following scenario suggests a natural application for Group GAP. Consider a Video-on-Demand (VoD) service where each video is given as a collection of segments. The system has a set of m servers of uniform capacity distributed over multiple locations. To obtain revenue from a video the system must store all of its segments (possibly on different servers). The revenue from a specific video also depends on the servers which store the segments. This is due to the content delivery costs resulting from the distance between the servers and the predicted location of the video audience. The objective of the VoD service provider is to select a subset of segments and an allocation of these segments to servers so as to maximize the total revenue. In [16] we describe central applications of Group GAP in mobile Device-to-Device network caching and in base station cooperation in 5G networks.

1.1 Prior Work

We note that a Group GAP instance in which each group consists of a *single* item yields an instance of classic GAP where each item takes a single size across the bins, and all the bins have identical capacities. GAP is known to be APX-hard already in this case, even if there are only two possible item sizes, and each item can take one of two possible profits [8]. Thus, most of the previous research focused on obtaining efficient approximate solutions.[1] Fleischer et al. [14] obtained a $(1 - e^{-1})$-approximation for GAP, as a special case of the *separable assignment problem.* Feige and Vondrák [12] obtained the current best known ratio of $1 - e^{-1} + \varepsilon$, for some absolute constant $\varepsilon > 0$.

[1] Given an algorithm $\mathcal{A}$, let $\mathcal{A}(I), OPT(I)$ denote the profit of the solution output by $\mathcal{A}$ and by an optimal solution for a problem instance I, respectively. For $\rho \in (0, 1]$, we say that $\mathcal{A}$ is a ρ-approximation algorithm if, for any instance I, $\frac{\mathcal{A}(I)}{OPT(I)} \geq \rho$.

Chen and Zhang [9] studied the problem of *group packing of items into multiple knapsacks* (GMKP), a special case of Group GAP where the profit of each item is the same across the bins. Let GMKP(δ) be the restriction of GMPK to instances in which the total size of items in each group is at most δm (that is, a factor δ of the total capacity of all bins). For $\delta > \frac{2}{3}$, the paper [9] rules out the existence of a constant factor approximation for GMKP(δ), unless P= NP. For $\frac{1}{3} < \delta \leq \frac{2}{3}$, the authors show that there is no $\left(\frac{1}{2} + \varepsilon\right)$-approximation for GMKP($\delta$), unless P= NP, and derive a nearly matching $\left(\frac{1}{2} - \varepsilon\right)$-approximation, for any $\varepsilon > 0$. The paper presents also approximation algorithms and hardness results for other special cases of GMPK.

There has been earlier work also on variants of Group GAP with the added constraint that in any feasible assignment there is at most one item from G_ℓ in bin j, for any $1 \leq \ell \leq L$, $j \in [m]$. Adany et al. [1] considered this problem, called *all-or-nothing* GAP (AGAP). They presented a $(\frac{1}{19} - \varepsilon)$-approximation algorithm for general instances, and a $(\frac{1}{3} - \varepsilon)$-approximation for the special case where the profit of an item is identical across the bins, called the *group packing* (GP) problem. Sarpatwar et al. [20] consider a more general setting for AGAP, where each group of items is associated with a time window in which it can be packed. The paper shows that this variant of the problem, called χ-AGAP, admits an $\Omega(1)$-approximation, assuming the time windows are large enough relative to group sizes. Specifically, for a group G_ℓ having a time window of m slots ($= m$ bins), it is assumed that $s(G_\ell) \leq \frac{m}{20}$.

1.1.1 Submodular Maximization

Given a finite set Ω, a function $f : 2^\Omega \to \mathbb{R}$ is *submodular* if for every $S, T \subseteq \Omega$ we have

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T).$$

An equivalent definition of submodularity refers to its diminishing returns: for any $T \subseteq S \subseteq \Omega$, and $u \in \Omega \setminus S$,

$$f(S \cup \{u\}) - f(S) \leq f(T \cup \{u\}) - f(T).$$

A set function f is *monotone* if for every $S \subseteq T \subseteq \Omega$ it holds that $f(S) \leq f(T)$. Submodular functions arise naturally in a wide variety of optimization problems, ranging from coverage problems and graph cut problems to welfare problems (see [6] for a survey on submodular functions). Submodular optimization under various constraints has been widely studied in the past four decades (see, e.g., [22, 7, 13] and [6] and the references therein).

The problem of maximizing a monotone submodular function subject to a knapsack constraint is defined as follows. We are given an oracle to a monotone, non-negative submodular function $f : 2^\Omega \to \mathbb{R}_{\geq 0}$. Each element $i \in \Omega$ is associated with a size $s_i \geq 0$. We are also given a capacity $B > 0$. The objective is to find a subset $S \subseteq \Omega$ such that $\sum_{i \in S} s_i \leq B$ and $f(S)$ is maximized. The best known result is a $(1 - e^{-1})$-approximation algorithm due to Sviridenko [22]. The ratio of $(1 - e^{-1})$ cannot be improved even when f is a coverage function and element sizes are uniform, unless P= NP [11]. A matching lower bound of $(1 - e^{-1})$ is known also for the oracle model with no complexity assumption [19].

1.2 Contribution and Techniques

Our main result is a $\frac{1}{6}$-approximation algorithm for Group GAP instances where the total size of each group is at most $\frac{m}{2}$. We note that when group sizes can be arbitrary in $(0, m]$, Group GAP cannot be approximated within any bounded ratio, even if item profits are identical across the bins, and $m = 2$, unless P= NP. Indeed, in this case, deciding whether a single group of items of total size 2 and total profit 1 can be packed in the bins yields an instance of Partition, which is NP-complete [15]. Furthermore, even if group sizes are restricted to be no greater than δm, for some $\delta > \frac{2}{3}$, then Group GAP still cannot be approximated within a constant factor, as it generalizes GMKP(δ), for which the paper [9] shows hardness of approximation. Similarly, as we consider in this paper a generalization of GMKP$\left(\frac{1}{2}\right)$, it follows from [9] that our problem cannot be approximated within ratio better than $\frac{1}{2}$.

In solving Group GAP we combine the framework of Adany et al. [1] with the rounding technique of Shmoys and Tardos [21]. The framework of [1] uses submodular maximization to select a collection of groups for the solution. It then finds a feasible assignment for the selected groups.

At the heart of our algorithm lies an interesting derivation of a submodular function from the classic LP formulation of GAP, which facilitates the construction of a high profit solution utilizing at most half the total bin capacity. In particular, we give an algorithm for submodular maximization subject to a knapsack constraint, which finds a solution occupying at most half the knapsack capacity, while the other half is *reserved* for later use.[2] We show that this algorithm achieves an approximation ratio of $\frac{1}{3}$ relative to an optimal solution that may use the whole knapsack capacity. We note that this ratio is tight. Indeed, it is easy to construct an instance for which the best solution with half the knapsack capacity has only $\frac{1}{3}$ the profit of the optimal solution with full knapsack capacity. We also note that a naive application of the algorithm of Sviridenko [22] with half the knapsack capacity will only guarantee a $\frac{1-e^{-1}}{3} \approx \frac{1}{4.7}$-approximation.

To obtain an integral solution, given a fractional assignment of the selected groups, we apply the rounding technique of Shmoys and Tardos [21], followed by a filling phase. We show that if the total size of the items in the selected groups is at most $\frac{m}{2}$, the rounding procedure yields a *feasible* assignment of the selected groups, whose profit is at least half the value of the submodular function. Our novel approach of submodular optimization subject to a knapsack *with reserved capacity* constraint may find applications in solving other group assignment problems.

2 Approximation Algorithm

In this section we present an approximation algorithm for Group GAP. We first introduce several definitions and tools that will be used as building blocks of our algorithm.

2.1 Basic Definitions and Tools

2.1.1 The Submodular Relaxation

For simplicity, we assume that all the numbers are rational. For a subset of elements $I' \subseteq I$, let $s(I') = \sum_{i \in I'} s_i$ be the total size of the elements in I'. We assume throughout the discussion that every $G_\ell \in \mathcal{G}$ satisfies $s(G_\ell) \leq \frac{m}{2}$. We define below a function $\phi : 2^I \rightarrow \mathbb{R}_{\geq 0}$. Let $x_{i,j} \in \{0, 1\}$ be an indicator for the assignment of item i to bin j, for $1 \leq i \leq n$, $1 \leq j \leq m$. The following is a linear program associated with a subset $S \subseteq I$, in which $x_{i,j} \geq 0$ for all i, j.

[2] We assume throughout the discussion that the size of each element is at most half the knapsack capacity.

$$
\begin{aligned}
LP(S)\text{: maximize} \quad & \sum_{i \in I, j \in M} x_{i,j} \cdot p_{i,j} \\
\text{subject to:} \quad & \sum_{j \in M} x_{i,j} \le 1 \qquad \forall i \in I && (1)\\
& \sum_{i \in I} x_{i,j} \cdot s_i \le 1 \quad \forall j \in M && (2)\\
& x_{i,j} = 0 \qquad\quad \forall i \in I \setminus S, j \in M \\
& x_{i,j} \ge 0 \qquad\quad \forall i \in I, j \in M
\end{aligned}
$$

Note that, by the above constraints, all solutions for the LP have the same dimension, regardless of the size of S. We define $\phi(S)$ as the optimal value of LP(S).

We denote the profit of a solution x for the linear program by $p \cdot x = \sum_{i \in I, j \in M} x_{i,j} \cdot p_{i,j}$. In Section 3 we prove the next result.

▶ **Theorem 1.** *The function ϕ is submodular.*

We note that ϕ is also monotone and non-negative. We use ϕ to define the *group function* $\psi : 2^{\mathcal{G}} \to \mathbb{R}_{\ge 0}$. For any $G^* \subseteq \mathcal{G}$ let $I(G^*) = \bigcup_{G_\ell \in G^*} G_\ell$ and $\psi(G^*) = \phi(I(G^*))$. As ϕ is submodular, monotone and non-negative, it is easy to see that ψ is submodular, monotone and non-negative as well. We optimize ψ subject to a knapsack (budget) constraint, using the next general result.

▶ **Theorem 2** (Submodular optimization with reserved capacity)**.** *Let $\Omega = \{1, \ldots, n\}$ be a ground set, and $m \ge 0$ a knapsack capacity. Each $i \in \Omega$ is associated with non-negative size $s_i \le \frac{m}{2}$. Let $f : 2^\Omega \to \mathbb{R}_{\ge 0}$ be a non-negative monotone submodular function, and $OPT = \max\{f(S) | S \subseteq \Omega, \sum_{i \in S} s_i \le m\}$. Then Algorithm 2 (in Section 4) finds in polynomial time[3] a subset $S \subseteq \Omega$ satisfying $f(S) \ge \frac{OPT}{3}$ and $\sum_{i \in S} s_i \le \frac{m}{2}$.*

The proof of Theorem 2 is given in Section 4.

2.1.2 Solution Types

Our algorithm uses a few types of intermediate solutions for Group GAP, as defined below. Given $G^* \subseteq \mathcal{G}$, we say that a solution x for $LP(I(G^*))$ is a *fractional solution*. Let $U = (U_1, \ldots, U_m)$ be an assignment of elements to bins, where U_j is the set of elements assigned to bin j. Then $I(U) = \bigcup_{j=1}^{m} U_j$ is the subset of elements packed in the bins. We say that U is *feasible* if for each bin $1 \le j \le m$ we have $s(U_j) \le 1$. We say that U is *almost feasible* if for each bin $1 \le j \le m$ there is an element u_j^* such that $s(U_j \setminus \{u_j^*\}) \le 1$. We also define the profit of an assignment as $p(U) = \sum_{1 \le j \le m} \sum_{i \in U_j} p_{i,j}$.

Our algorithm first obtains a fractional solution, which is then converted to an almost feasible solution. Finally, the algorithm converts this solution to a feasible one. We now state the results used in these conversion steps.

[3] The explicit representation of a submodular function might be exponential in the size of its ground set. Thus, it is standard practice to assume that the function is accessed via a value oracle. Then the number of operations and oracle calls is polynomial in the size of Ω and the maximum length of the representation of $f(S)$.

▶ **Theorem 3.** *Given $G^* \subseteq \mathcal{G}$, such that $s(I(G^*)) \leq m$, and a fractional solution x for $LP(I(G^*))$, it is possible to construct in polynomial time an almost feasible assignment U such that $p(U) \geq p \cdot x$, and $I(U) = I(G^*)$.*

The theorem easily follows by applying a rounding technique of [21] to a fractional solution in which every element in $I(G^*)$ is fully assigned (fractionally, in multiple bins). We note that such a solution always exists, since $s(I(G^*)) \leq m$. We give the proof in the full version of the paper [16]. To convert an almost feasible solution to a feasible one we use the following result (we give the proof in Section 5).

▶ **Theorem 4.** *Let $U = (U_1, \dots, U_m)$ be an almost feasible assignment such that $s(I(U)) \leq \frac{m}{2}$, then U can be converted in polynomial time to a feasible assignment U', with $I(U') = I(U)$ and $p(U') \geq \frac{1}{2}p(U)$.*

2.2 The Algorithm

Our approximation algorithm for Group GAP follows easily from the tools presented in Section 2.1. Initially, we solve the problem of maximizing a submodular function subject to a knapsack with reserved capacity constraint for the set function ψ. Then we solve the linear program and convert the solution to a feasible assignment. We give the pseudocode in Algorithm 1.

Algorithm 1 Group GAP Algorithm.

1: Solve the submodular optimization problem: $\max_{\{S \subseteq \mathcal{G}, \sum_{G_\ell \in S} s(G_\ell) \leq m/2\}} \psi(S)$ using Algorithm 2. Let S^* be the solution found.
2: Find a (fractional) solution x for $LP(I(S^*))$ that realizes $\psi(S^*)$.
3: Use Theorem 3 to convert x to an almost feasible assignment U with $I(U) = I(S^*)$.
4: Use Theorem 4 to convert U into a feasible solution; return this solution.

▶ **Theorem 5.** *Algorithm 1 is a polynomial time $\frac{1}{6}$-approximation algorithm for Group GAP when the total size of a single group is bounded by $\frac{m}{2}$; that is, $\forall G_\ell \in \mathcal{G} : \sum_{i \in G_\ell} s_i \leq \frac{m}{2}$.*

Proof. It is easy to see that the algorithm runs in polynomial time. By Theorem 2, we have that $\psi(S^*) \geq \text{OPT}/3$, where OPT is the value of the optimal solution for the original instance.

By Theorems 3 and 4, we are guaranteed to find in Steps 3−4 a feasible assignment U of all elements in $I(S^*)$, such that $p(U) \geq \frac{1}{2}\psi(S^*) \geq \frac{1}{6}\text{OPT}$. ◀

3 Submodularity

In this section we show that the function ϕ is submodular. Our proof builds on the useful relation of our problem to maximum weight bipartite matching. Let $G = (A \cup B, E)$ be a bipartite (edge) weighted graph, where $|B| \geq |A|$. Assume that the graph is complete (by adding zero weight edges if needed). For $e \in E$, let $W(e)$ be the weight of edge e, and for $F \subseteq E$, let $W(F) = \sum_{e \in F} W(e)$ be the total weight of edges in F. For $S \subseteq A$, define $h(S)$ to be the value of the maximum weight matching in $G[S \cup B]$, the graph induced by $S \cup B$. We call h the *partial maximum weight matching function.* The next result was shown by Bar-Noy and Rabanca [3].

▶ **Theorem 6.** *If the edge weights are non-negative then the function h is (monotone) submodular.*

We give a simpler proof in the full version of the paper [16]. We are now ready to prove our main result.

Proof of Theorem 1. We first note that since all numbers are rational, for some $N \in \mathbb{Z}^+$, we can write $s_i = \frac{\hat{s}_i}{N}$, where $\hat{s}_i \in \mathbb{Z}^+$ for all $i \in I$.[4]

Now, set the capacity of each bin $1 \leq j \leq m$ to be $b_j = N$, and let $0 \leq y_{i,j} \leq \hat{s}_i$ indicate the size of item i assigned to bin j. For a subset of items $S \subseteq I$, we now write the following linear program.

$$
\begin{aligned}
M(S)\text{: maximize} \quad & \sum_{i \in I} \frac{1}{\hat{s}_i} \sum_{j \in M} y_{i,j} \cdot p_{i,j} & & \\
\text{subject to:} \quad & \sum_{j \in M} y_{i,j} \leq \hat{s}_i & \forall i \in I & \qquad (3) \\
& \sum_{i \in I} y_{i,j} \leq N & \forall j \in M & \qquad (4) \\
& y_{i,j} = 0 & \forall i \in I \setminus S, j \in M & \\
& y_{i,j} \geq 0 & \forall i \in I, j \in M &
\end{aligned}
$$

Indeed, Constraint (3) ensures that the total size assigned for item i over the bins is upper bounded by $\hat{s}_i$, and Constraint (4) guarantees that the capacity constraint is satisfied for all the bins $j \in M$. Given a subset of elements $S \subseteq I$, let $\eta(S)$ be the value of an optimal solution for $M(S)$.

Now, observe that any feasible solution for $LP(S)$ induces a feasible solution for $M(S)$ of the same value, by setting $y_{i,j} = x_{i,j} \cdot \hat{s}_i$ for all $i \in I$ and $j \in M$. Similarly, a feasible solution for $M(S)$ induces a feasible solution for $LP(S)$ of the same value. Hence, $\phi(S) = \eta(S)$ for all $S \subseteq I$.

By the above discussion, to prove the theorem it suffices to show that η is submodular. Given our Group GAP instance, we construct the following bipartite graph G. For each item $i \in I$, we define $\hat{s}_i$ vertices, $V_i = \{v_{i,1}, \ldots, v_{i,\hat{s}_i}\}$. For each bin $j \in M$, we define N vertices $U_j = \{u_{j,1}, \ldots, u_{j,N}\}$. For any $i \in [n]$ and $j \in [m]$, there are edges $(v_{i,s}, u_{j,r})$ of weight $p_{i,j}/\hat{s}_i$, for all $1 \leq s \leq \hat{s}_i$, $1 \leq r \leq N$. Let $V_I = \cup_{i \in I} V_i$, $U_M = \cup_{j \in M} U_j$, and let E be the set of edges. Consider the bipartite graph $G = (V_I \cup U_M, E)$. W.l.o.g we may assume that $|U_M| \geq |V_I|$; otherwise, we can add new bins $j = m+1, m+2, \ldots$ with corresponding sets of vertices $U_j = \{u_{j,1}, \ldots, u_{j,N}\}$ and zero weight edges $(v_{i,s}, u_{j,r})$ for all $i \in [n]$, $1 \leq s \leq \hat{s}_i$, $1 \leq r \leq N$.

We note that, given a subset of items $S \subseteq I$, $M(S)$ is the linear programming relaxation of the problem of finding a maximum weight matching in the subgraph $G[V_S \cup U_M]$, where $V_S \subseteq V_I$ is the subset of vertices in G that corresponds to S. Using standard techniques (see, e.g., [17]), it can be shown that $M(S)$ has an optimal integral solution. Hence, $\eta(S) = h(V_S)$, where $h : 2^{V_I} \to \mathbb{R}_{\geq 0}$ is a partial maximum weight matching function in G. By Theorem 6, h is (monotone) submodular. Hence, η is also (monotone) submodular. ◀

[4] Note that N, which may be arbitrarily large, is used just for the proof. Our algorithm does not rely on obtaining a solution (or an explicit formulation) for $M(S)$.

4 Submodular Optimization with Reserved Capacity

In this section we prove Theorem 2. We start with some definitions and notation. Assume we are given a ground set $\Omega = \{1, \dots, n\}$ and capacity $m > 0$, where each element $i \in \Omega$ is associated with a non-negative size $s_i \le \frac{m}{2}$. For $S \subseteq \Omega$, let $s(S) = \sum_{i \in S} s_i$. Also, for $S, T \subseteq \Omega$ let $f_S(T) = f(S \cup T) - f(S)$. We use throughout this section basic properties of monotone submodular functions (see, e.g., [6]).

Algorithm 2 SubmodularOpt.

Input: A monotone submodular function $f : 2^{\Omega} \to \mathbb{R}_{\ge 0}$, sizes $s_i \ge 0$ for all $i \in \Omega$, and capacity $m > 0$.
Output: A subset of elements $R \subseteq \Omega$ such that $s(R) \le \frac{m}{2}$.

procedure Greedy(g, m')
  Set $S = \emptyset$, $E = \Omega$.
  while $E \setminus S \ne \emptyset$ **do**
    Find $i = \arg\max_{i \in E \setminus S} \frac{g_S(\{i\})}{s_i}$
    if $s(S) + s_i \le m'$ **then** set $S = S \cup \{i\}$.
    end if
    Set $E = E \setminus \{i\}$.
  end while
  Return S
end procedure
Set $R = \emptyset$
for every set $S_e \subseteq \Omega$, $|S_e| \le 6$ **do**
  for every set $B \subseteq S_e$, $s(B) \le m/2$ **do**
    $T = \text{Greedy}(f_{S_e}, m/2 - s(B))$
    if $f(B \cup T) \ge f(R)$ **then** Set $R = B \cup T$.
    end if
  end for
end for
Return R

In the following we give an outline of an algorithm for maximizing a monotone submodular function f subject to a knapsack *with reserved capacity* constraint. Specifically, assuming that the knapsack capacity is m for some $m > 0$, the algorithm solves the problem $\max_{\{S \subseteq \Omega : s(S) \le \frac{m}{2}\}} f(S)$. The algorithm, SubmodularOpt, initially guesses the set of at most six items of highest profits in some optimal solution (for the problem with knapsack capacity m), and a subset of these profitable items, whose total size is at most $m/2$. Then the algorithm calls a procedure which applies the Greedy approach as in [22] to find the remaining items in the solution. We give a pseudocode of SubmodularOpt in Algorithm 2. It is important to note that while the algorithm produces a solution of size at most $m/2$, the analysis compares this solution against an optimal solution of size at most m.

The next lemma, which plays a key role in our analysis, follows from the technique presented in [22].

► **Lemma 7.** *Given the knapsack capacity $m > 0$, let $0 < m' \le m^* \le m$. Let $S^* \subseteq \Omega$ be a non-empty subset of elements, such that $s(S^*) \le m^*$. Also, let $g : 2^{\Omega} \to \mathbb{R}$ be a monotone submodular function satisfying $g(\emptyset) = 0$, and let $S = \text{Greedy}(g, m')$. Then, there is an element $i^* \in S^*$ such that $g(S) + g(\{i^*\}) \ge (1 - e^{-m'/m^*}) g(S^*)$.*

In the full version of the paper [16] we prove a more general result (see Lemma 12 therein). Lemma 7 is obtained by setting $T = \emptyset$ in Lemma 12 in [16].

Lemma 7 was applied in [22] in the special case where $m' = m^*$. It was applied in conjunction with a guessing phase, used to ensure that the three most profitable elements in an optimal solution are selected by the algorithm, thus bounding the value of $g(\{i^*\})$.

Several difficulties arise while attempting to apply a similar approach to the problem with *reserved* capacity. The first one is that the most profitable elements in *any* optimal solution may already exceed the reduced capacity, and therefore cannot be added to the solution. Another difficulty is that even if these elements do fit in the smaller knapsack, one can easily come up with a scenario in which it is better not to include them in the solution.

To overcome these difficulties we use the following main observation. Given P_k, the set of $k = 6$ most profitable elements in an optimal solution[5], and a partition of this set into two subsets B_1, B_2, each of size at most $m/2$ (if such a partition exists), adding elements to either B_1 or B_2 using the greedy procedure leads to a solution of value at least one third of the value of an optimal solution. This observation comes into play in Case 2.2 in the proof of Theorem 2. The next technical lemma is used to prove this observation (we give the proof below).

▶ **Lemma 8.** *For $k = 6$, $p_A, p_B, S_A, S_B \geq 0$ such that $p_A + p_B \leq 1$ and $S_A + S_B \leq 1$, define*

$$h(p_A, p_B, S_A, S_B) = p_A + (1 - p_A - p_B)\left(1 - e^{-\frac{\frac{1}{2} - S_A}{1 - S_A - S_B}}\right) - \frac{p_A + p_B}{k}.$$

Then for p_1, p_2, S_1, S_2 such that $0 \leq p_1, p_2 \leq \frac{1}{3}$ and $0 \leq S_1, S_2 \leq \frac{1}{2}$ it holds that

$$\max\left(h(p_1, p_2, S_1, S_2), h(p_2, p_1, S_2, S_1)\right) \geq \frac{1}{3}.$$

Another main tool used in the proof of Theorem 2 is a simple partitioning procedure. It shows that P_k can either be partitioned into two sets as required in the above observation, or we reach a simple corner case (Case 2.1 in the proof) in which at least one third of the optimal value can be easily attained. For the latter case, we use the following result, due to [18].

▶ **Lemma 9.** *Let $g : 2^\Omega \to \mathbb{R}$ be a non-negative and monotone submodular function. Let $OPT = \max\{g(S) | S \subseteq \Omega, \sum_{i \in S} s_i \leq m\}$, and $S^* = \textsc{Greedy}(g, m)$. Then either $g(S^*) \geq (1 - e^{-1/2})OPT$, or there is an element $i \in \Omega$ such that $g(\{i\}) \geq (1 - e^{-1/2})OPT$ and $s_i \leq m$.*

Proof of Theorem 2. (Submodular optimization with reserved capacity). It is easy to see that the running time of the algorithm is polynomial. Let $S \subseteq \Omega$, $s(S) \leq m$, $f(S) = \text{OPT}$, and $k = 6$.

Case 1: We first handle the case where $|S| < k$. We prove that in this case the algorithm finds a set R such that $f(R) \geq \text{OPT}/3$. Start with $A_1 = \emptyset$, iterate over the elements of S and add them to A_1, as long as $s(A_1) \leq m/2$. If $S \neq A_1$, let $j \in S \setminus A_1$, and set $A_2 = \{j\}$ and $A_3 = S \setminus (A_1 \cup A_2)$. Clearly, $s(A_2) \leq m/2$, and since $s(A_1 \cup A_2) > m/2$ and $s(S) \leq m$, we have that $s(A_3) \leq m/2$. If $S = A_1$ set $A_2 = A_3 = \emptyset$.

[5] The value $k = 6$ is derived from Lemma 8. It may be possible to obtain the same approximation ratio using smaller values of k, leading to a more efficient algorithm.

By the submodularity of f, we have $f(S) \leq f(A_1) + f(A_2) + f(A_3)$. Hence, for some $r \in \{1, 2, 3\}$, $f(A_r) \geq f(S)/3 = \text{OPT}/3$. We also have that $|A_r| \leq 5$; therefore, at some iteration of the algorithm $S_e = B = A_r$, and following this iteration $f(R) \geq \text{OPT}/3$.

Case 2: Assume now that $|S| \geq k$. Let $S = \{i_1, i_2, \ldots, i_\ell\}$ such that the elements are ordered by their marginal profits: $i_j = \arg\max_{j \leq r \leq \ell} f_{\{i_1, \ldots i_{j-1}\}}(\{i_r\})$. Set $P_k = \{i_1, i_2, \ldots, i_k\}$. Consider the following process. Start with $B_1 = \emptyset$ and $B_2 = \emptyset$. Iterate over the elements $i \in P_k$ in decreasing order by size. For each element i, let $t = \arg\min_{j=1,2} s(B_j)$. If $s(B_t) + s_i \leq m/2$ then $B_t = B_t \cup \{i\}$; otherwise, Stop. We now distinguish between two sub-cases for the termination of the process.

Case 2.1: Suppose that the process terminates due to an element i which cannot be added to any of the sets. Let B_1 and B_2 be the sets in this iteration. Also, set $B_3 = \{i\}$, $U = B_1 \cup B_2 \cup B_3$, and $L = S \setminus U$. W.l.o.g assume that $s(B_1) \geq s(B_2)$. As the process terminated, we have that $s(B_3) + s(B_2) = s_i + s(B_2) > m/2$. The sets B_1, B_2, B_3 and L form a partition of S, and $s(S) \leq m$. We conclude that $s(B_1) + s(L) \leq m/2$. Hence, $s(B_2) + s(L) \leq m/2$, and $s(B_3) + s(L) \leq m/2$ as well (it is easy to see that $s(B_3) \leq s(B_1)$). By the submodularity of f, $f(U) \leq f(B_1) + f(B_2) + f(B_3)$; thus, there is $j \in \{1, 2, 3\}$ such that $f(B_j) \geq f(U)/3$. As none of the sets B_1, B_2, B_3 is empty, we have that $|B_j| \leq |P_k| - 2 = 4$.

Let $T = \textsc{Greedy}(f_U, m/2 - s(B_j))$. By Lemma 9, either

$$f_U(T) \geq (1 - e^{-1/2}) f_U(L) \geq f_U(L)/3,$$

or there is $i \in L$ such that

$$f_U(\{i\}) \geq (1 - e^{-1/2}) f_U(L) \geq f_U(L)/3.$$

In the former case, we can consider the iteration in which $S_e = U, B = B_j$. In this iteration, we have

$$f(B \cup T) \geq f(B_j) + f_U(T) \geq \frac{1}{3}(f(U) + f_U(L)) = \frac{1}{3}\text{OPT}.$$

In the latter case, we can consider the iteration where $S_e = B = B_j \cup \{i\}$, and in which

$$f(B \cup T) \geq f(B) \geq f(B_j) + f_{B_j}(\{i\}) \geq f(B_j) + f_U(\{i\}) \geq \frac{1}{3}\text{OPT}.$$

Case 2.2: The process terminated with B_1, B_2 satisfying $B_1 \cup B_2 = P_k$, and $s(B_1), s(B_2) \leq m/2$.

Let $p_1 = f(B_1)/\text{OPT}$, $p_2 = (f(P_k) - f(B_1))/\text{OPT}$, $S_1 = s(B_1)$, $S_2 = s(B_2)$ and $L = S \setminus P_k$. If $p_1 \geq \frac{1}{3}$ (or $p_2 \geq \frac{1}{3}$) we have that in the iteration where $S_e = P_k$ and $B = B_1$ (or $B = B_2$) the algorithm finds a solution of value at least $\text{OPT}/3$, and the theorem holds. Thus, we may assume that $p_1, p_2 \leq \frac{1}{3}$.

For $j = 1, 2$, let $T_j = \textsc{Greedy}(f_{P_k}, m/2 - S_j)$. Using Lemma 7 with $S^* = L$, we have that there is $i_j \in L$ for which

$$f_{P_k}(T_j) + f_{P_k}(\{i_j\}) \geq (1 - e^{-\frac{\frac{1}{2} - S_j}{1 - S_1 - S_2}}) f_{P_k}(L).$$

By the selection of elements in P_k, we have $f_{P_k}(\{i_j\}) \leq \frac{1}{k} f(P_k)$. Thus,

$$f_{B_j}(T_j) \geq f_{P_k}(T_j) \geq (1 - e^{-\frac{\frac{1}{2} - S_j}{1 - S_1 - S_2}}) f_{P_k}(L) - \frac{1}{k} f(P_k).$$

Hence, in the iteration where $S_e = P_k, B = B_j$, we obtain a solution satisfying

$$f(S_e \cup T) = f(B_j \cup T_j) \geq f(B_j) + (1 - e^{-\frac{\frac{1}{2} - S_j}{1 - S_1 - S_2}}) f_{P_k}(L) - \frac{1}{k} f(P_k)$$
$$\geq \text{OPT}\left(p_j + (1 - p_1 + p_2)(1 - e^{-\frac{\frac{1}{2} - S_j}{1 - S_1 - S_2}}) - \frac{p_1 + p_2}{k}\right).$$

By Lemma 8, in one of these iterations we obtain a solution of value at least OPT/3, implying the statement of the theorem. ◀

Proof of Lemma 8. Let p_1, p_2, S_1, S_2 be values that satisfy the conditions in the lemma. Denote $p = p_1 + p_2$, $d = p_1 - p_2$ and $r = e^{-\frac{\frac{1}{2} - S_1}{1 - S_1 - S_2}}$. It is easy to see that $e^{-\frac{\frac{1}{2} - S_2}{1 - S_1 - S_2}} = e^{-1} r^{-1}$. Define $V_1 = h(p_1, p_2, S_1, S_2)$ and $V_2 = h(p_2, p_1, S_2, S_1)$. By the definition of h and above definitions we get

$$V_1 = \frac{p + d}{2} + (1 - p)(1 - r) - \frac{p}{k}$$

and

$$V_2 = \frac{p - d}{2} + (1 - p)\left(1 - e^{-1} r^{-1}\right) - \frac{p}{k}$$

Let $g_1(x) = \frac{p+d}{2} + (1 - p)(1 - x) - \frac{p}{k}$ and $g_2(x) = \frac{p-d}{2} + (1 - p)\left(1 - e^{-1} x^{-1}\right) - \frac{p}{k}$. Clearly, $V_1 = g_1(r)$ and $V_2 = g_2(r)$. It is also easy to see that g_1 is decreasing and g_2 is increasing (for $x > 0$).

▷ Claim 10. It holds that $g_1(x^*) = g_2(x^*)$ where $x^* = \frac{d + \sqrt{d^2 + 4e^{-1}(1-p)^2}}{2(1-p)}$.

Proof. By rearranging terms we have $g_1(x) = g_2(x)$ if and only if $d = (1 - p)(x - e^{-1} x^{-1})$, which holds for $x > 0$ if and only if $0 = (1 - p)x^2 - dx - e^{-1}(1 - p)$. The latter is a quadratic equation and $x^* > 0$ is a root. ◁

If $r \geq x^*$, since g_2 is increasing, we have $V_2 = g_2(r) \geq g_2(x^*) = g_1(x^*)$, and if $r \leq x^*$, as g_1 is decreasing, we have $V_1 = g_1(r) \geq g_1(x^*)$. Therefore $\max(V_1, V_2) \geq g_1(x^*)$. By rearranging terms and substituting $k = 6$ we have

$$g_1(x^*) = 1 - \frac{p}{2} - \frac{\sqrt{d^2 + 4e^{-1}(1 - p)^2}}{2} - \frac{p}{6} = 1 - \frac{2p}{3} - \frac{\sqrt{d^2 + 4e^{-1}(1 - p)^2}}{2}$$

Our goal is to show that for $0 \leq p \leq \frac{2}{3}$, we have $1 - \frac{2p}{3} - \frac{1}{2} \cdot \sqrt{d^2 + 4e^{-1}(1 - p)^2} \geq \frac{1}{3}$, or $\frac{2}{3}(1 - p) - \frac{1}{2} \cdot \sqrt{d^2 + 4e^{-1}(1 - p)^2} \geq 0$. By rearranging terms this is equivalent to showing

$$\frac{2}{3} \geq \sqrt{\frac{d^2}{4(1 - p)^2} + e^{-1}}.$$

Consider two cases:
Case 1: $0 \leq p \leq \frac{1}{3}$. Since $|d| \leq p$ we have

$$\sqrt{\frac{d^2}{4(1 - p)^2} + e^{-1}} \leq \sqrt{\frac{p^2}{4(1 - p)^2} + e^{-1}} \leq \sqrt{\frac{1}{16} + e^{-1}} \leq \frac{2}{3}.$$

The last inequality follows since the function $\frac{x}{1-x}$ is increasing in the interval $[0, \frac{1}{3}]$.

Case 2: $\frac{1}{3} \leq p \leq \frac{2}{3}$, as $\frac{p+d}{2}, \frac{p-d}{2} \leq \frac{1}{3}$, we have that $|d| \leq \frac{2}{3} - p$. Therefore,

$$\sqrt{\frac{d^2}{4(1-p)^2} + e^{-1}} \leq \sqrt{\frac{(2/3-p)^2}{4(1-p)^2} + e^{-1}} \leq \sqrt{\frac{1}{16} + e^{-1}} \leq \frac{2}{3}.$$

The last inequality follows since the function $\frac{2/3-x}{1-x}$ is decreasing in the interval $[\frac{1}{3}, \frac{2}{3}]$.

In both cases we get $g_1(x^*) \geq \frac{1}{3}$, and as $\max(V_1, V_2) \geq g_1(x^*)$, the lemma follows. ◀

5 Filling Phase

In this section we prove Theorem 4. Define the size of bin j in assignment U as $s_j^U = \sum_{i \in U_j} s_i$. We first divide the bins and items into types. We say that a bin j is *full* if $s_j^U > 1$, *semi-full* if $\frac{1}{2} \leq s_j^U \leq 1$, and *semi-vacant* if $s_j^U < \frac{1}{2}$. An item $i \in I(U)$ is *big* if $s_i > \frac{1}{2}$; otherwise, i is *small*. Clearly, there are no big items in semi-vacant bins.

Informally, we use in the proof several types of resolution steps. Each step takes as input a full bin and possibly one or two semi-vacant bins, and reassigns some of the items into the bins while evicting others. These resolution steps ensure that the new assignment has at least half the profit of the original assignment, the assignment to any bin remains feasible, and only small items are evicted.

We apply the resolution steps repeatedly, but once a bin participated in a resolution step it may not participate in another one. We then prove that as long as there are full bins, one of the steps can be applied. Hence, by applying the resolution steps, we have a new assignment in which all bins are feasible, and the total profit is at least half the profit of the original assignment. To handle the evicted items, we note that as $s(I(U)) \leq m/2$ and all the evicted items are small, it is possible to assign the evicted items to bins without violating the capacity constraints.

Proof of Theorem 4. For any bin $1 \leq j \leq m$ and $A \subseteq I$, let $p_j(A) = \sum_{i \in A} p_{i,j}$ be the total profit gained from packing A into j. The first step is to resolve the violation of the capacity constraint in each full bin. We do that using four types of resolution steps. Each step takes a full bin and possibly one or two semi-vacant bins, modifies their contents and adds some small elements to a set V of *evicted* elements, that will be handled later. Throughout the discussion, we consider for a full bin j a partition of the elements in U_j into two feasible subsets, given by $\{A_j, B_j\}$. We use the following resolution steps.

1. Consider a full bin j such that U_j has no big elements. If $p_j(A_j) > p_j(B_j)$ then set $U'_j = A_j$ and evict B_j ($V := V \cup B_j$); otherwise, set $U'_j = B_j$ and evict A_j, ($V := V \cup A_j$). In both cases U'_j is feasible and $p_j(U'_j) \geq \frac{1}{2} \cdot p_j(U_j)$.
2. Now, suppose we have a full bin j such that U_j has a single big element, and a semi-vacant bin ℓ. Let $\{A^*, B^*\} = \{A_j, B_j\}$, such that the big element is in A^*.
   If $p_j(A^*) + p_\ell(U_\ell) > p_j(B^*)$, set $U'_j = A^*$, $U'_\ell = U_\ell$ and evict the elements in B^* ($V := V \cup B^*$). We note that in this case $p_j(U'_j) + p_\ell(U'_\ell) = p_j(A^*) + p_\ell(U_\ell)$.
   Otherwise, set $U'_j = B^*$ and $U'_\ell = A^*$, and evict all the elements in U_ℓ, ($V := V \cup U_\ell$). In this case we have $p_j(U'_j) + p_\ell(U'_\ell) \geq p_j(B^*)$.
   Therefore, in both cases have $p_j(U'_j) + p_\ell(U'_\ell) \geq \frac{1}{2} \cdot (p_j(U_j) + p_\ell(U_\ell))$.
3. Consider a full bin j such that U_j has two big elements, and a semi-vacant bin ℓ, such that one of the big elements has space in bin ℓ; that is, there is a big element $i^* \in U_j$ such that $s_{i^*} + s_\ell^U \leq 1$.

Let $\{A^*, B^*\} = \{A_j, B_j\}$ such that $i^* \in A^*$. We note that there cannot be any other big element in A^* other that i^*.
If $p_j(B^*) + p_\ell(U_\ell) > p_j(A^*)$ set $U'_j = B^*$ and $U'_\ell = U_\ell \cup \{i^*\}$ (note that $s_\ell^{U'} \leq 1$). Also, evict all elements in $A^* \setminus \{i^*\}$. In this case we have $p_j(U'_j) + p_\ell(U'_\ell) \geq p_j(B^*) + p_\ell(U_\ell)$.
Otherwise, we set $U'_j = A^*$ and $U'_\ell = B^*$, and evict U_ℓ ($V := V \cup U_\ell$). In this case we have $p_j(U'_j) + p_\ell(U'_\ell) \geq p_j(A^*)$.
Thus, in both cases $p_j(U'_j) + p_\ell(U'_\ell) \geq \frac{1}{2} \cdot (p_j(U_j) + p_\ell(U_\ell))$.

4. Finally, consider a full bin j, such that U_j has two big elements, and two semi-vacant bins ℓ_1 and ℓ_2. Recall A_j, B_j is a partition of the elements in U_j into two feasible subsets. If $p_j(A_j) + p_{\ell_1}(U_{\ell_1}) > p_j(B_j) + p_{\ell_2}(U_{\ell_2})$, set $U'_j = A_j$, $U'_{\ell_1} = U_{\ell_1}$ and $U'_{\ell_2} = B_j$, and evict U_{ℓ_2}. Thus, we have

$$p_j(U'_j) + p_{\ell_1}(U'_{\ell_1}) + p_{\ell_2}(U'_{\ell_2}) \geq p_j(A_j) + p_{\ell_1}(U_{\ell_1}).$$

Otherwise, set $U'_j = B_j$, $U'_{\ell_1} = A_j$ and $U'_{\ell_2} = U_{\ell_2}$ and evict U_{ℓ_1}. Here

$$p_j(U'_j) + p_{\ell_1}(U'_{\ell_1}) + p_{\ell_2}(U'_{\ell_2}) \geq p_j(B_j) + p_{\ell_2}(U_{\ell_2}).$$

And finally, we always have

$$p_j(U'_j) + p_{\ell_1}(U'_{\ell_1}) + p_{\ell_2}(U'_{\ell_2}) \geq \frac{1}{2} \cdot (p_j(U_j) + p_{\ell_1}(U_{\ell_1}) + p_{\ell_2}(U_{\ell_2})).$$

Each time we execute a step we mark the bins used in this step as *resolved*, and we do not consider them in the next steps. We first use Steps 1, 2, and 3, until none of them can be applied.

Consider the average size of the unresolved bins. When we start, there are m bins of average size no greater than half. Each of Steps 1, 2, and 3 reduces the size of the unresolved bins by at least one (as a full bin is removed) and reduces the number of bins by at most two. Therefore, the average size of the unresolved bins remains no more than half. Also, marking all the semi-full bins as resolved will preserve the property.

Let a be the number of unresolved full bins and c be the number of unresolved semi-vacant bins. Due to the average size of bins, we have $a \leq \frac{a+c}{2}$, therefore $a \leq c$. Hence, if we have a full bin, there must be a semi-vacant bin as well. As we used Steps 1, 2, and 3 to exhaustion, every full bin must have two big elements (no bin in U contains more than two big elements), and none of these big elements can fit into one of the semi-vacant bins.

Denote the minimal size of a semi-vacant unresolved bin by r. Then each of the full bins has two big elements of size greater than $1 - r$. Hence, we have $c \cdot r + 2a \cdot (1 - r) < \frac{a+c}{2}$, which leads to $2a(1 - r) - \frac{a}{2} < c\left(\frac{1}{2} - r\right)$, and

$$c > a \cdot \frac{2 - 2r - \frac{1}{2}}{\frac{1}{2} - r} = a \cdot \frac{3 - 4r}{1 - 2r} = a \cdot \left(\frac{2 - 4r}{1 - 2r} + \frac{1}{1 - 2r}\right) = a \cdot \left(2 + \frac{1}{1 - 2r}\right) > 2a,$$

implying that we can now run Step 4, until there are no more unresolved full bins.

We use the resolution steps to eliminate all the full bins. Every time we run such a step over a set of bins we lose at most half the profit of the bins participating in the step. As the total size of items in the assignment is bounded by $m/2$, we are guaranteed that it is possible to assign all the evicted elements to some bins. Thus, we are able to resolve the capacity overflow while losing at most half of the profit. ◀

6 Discussion and Future Work

In this paper we presented a $\frac{1}{6}$-approximation algorithm for Group GAP, using a mild assumption on the size of each group. A key component in our result is an algorithm for submodular maximization subject to a knapsack constraint, which finds a solution occupying at most half the knapsack capacity, while the other half is *reserved* for later use. Our results leave several avenues for future work.

As mentioned above, Group GAP with no assumption on group sizes cannot be approximated within any constant factor. Yet, the maximum group size that still allows to obtain a constant ratio can be anywhere in $[\frac{m}{2}, \frac{2}{3}m]$. Thus, a natural question is: “Can our results be applied to instances with larger group sizes?” We note that the ratio stated in Theorem 5 may not hold already for instances in which group sizes can be at most $\frac{m}{2}(1+\varepsilon)$, for some $\varepsilon > 0$. Indeed, for such instances, it may be the case that no set of groups of total size at most $m/2$ is “good” relative to the optimum. The existence of an algorithm that yields a constant ratio for such instances remains open.

While our result for submodular optimization with reserved capacity (Theorem 2) gives an optimal approximation ratio for the studied subclass of instances, we believe the result can be extended to other subclasses. In particular, we conjecture that for instances where each item has size at most $\delta > 0$, the approximation ratio approaches $1 - e^{-\frac{1}{2}}$ as $\delta \to 0$. Such result would immediately imply an improved approximation ratio for instances of Group GAP in which the total size of each group is bounded by δm. We defer this line of work to the full version of the paper.

Lastly, we introduced in the paper the novel approach of submodular optimization subject to a knapsack with reserved capacity constraint. We applied the approach along with a framework similar to the one developed in [1]. It would be interesting to investigate whether the approach can be used to improve the approximation ratio obtained in [1] for *all-or-nothing* GAP.

References

1. Ron Adany, Moran Feldman, Elad Haramaty, Rohit Khandekar, Baruch Schieber, Roy Schwartz, Hadas Shachnai, and Tami Tamir. All-Or-Nothing Generalized Assignment with Application to Scheduling Advertising Campaigns. *ACM Trans. Algorithms*, 12(3):38:1–38:25, 2016.
2. V. Balachandran. An Integer Generalized Transportation Model for Optimal Job Assignment in Computer Networks. *Operations Research*, 24(4):742–759, 1976.
3. Amotz Bar-Noy and George Rabanca. Tight approximation bounds for the seminar assignment problem. In *International Workshop on Approximation and Online Algorithms*, pages 170–182. Springer, 2016.
4. Marco Bender, Clemens Thielen, and Stephan Westphal. Packing items into several bins facilitates approximating the separable assignment problem. *Inf. Processing Letters*, 115(6-8):570–575, 2015.
5. Niv Buchbinder and Moran Feldman. Deterministic Algorithms for Submodular Maximization Problems. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 392–403, 2016.
6. Niv Buchbinder and Moran Feldman. Submodular Functions Maximization Problems – A Survey. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics (2nd Edition)*, volume 1, chapter 42. Chapman and Hall/CRC, 2018.
7. Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a Monotone Submodular Function Subject to a Matroid Constraint. *SIAM J. on Computing*, 40(6):1740–1766, 2011.

8 C. Chekuri and S. Khanna. A PTAS for the Multiple Knapsack Problem. *SIAM J. on Computing*, 35(3):713–728, 2006.
9 Lin Chen and Guochuan Zhang. Packing Groups of Items into Multiple Knapsacks. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 28:1–28:13, 2016.
10 Robert G. Cromley and Dean M. Hanink. Coupling land use allocation models with raster GIS. *Journal of Geographical Systems*, 1(2):137–153, 1999.
11 Uriel Feige. A Threshold of $\ln n$ for Approximating Set Cover. *J. ACM*, 45(4), July 1998.
12 Uriel Feige and Jan Vondrák. Approximation algorithms for allocation problems: Improving the factor of 1-1/e. In *47th Annual IEEE Symposium on Foundations of Computer Science, FOCS'06*, pages 667–676, 2006.
13 M. Feldman, J. Naor, and R. Schwartz. A Unified Continuous Greedy Algorithm for Submodular Maximization. In *52nd Annual IEEE Symposium on Foundations of Computer Science, FOCS'11*, pages 570–579, 2011.
14 Lisa Fleischer, Michel X Goemans, Vahab S Mirrokni, and Maxim Sviridenko. Tight Approximation Algorithms for Maximum Separable Assignment Problems. *Math. Oper. Res.*, 36(3):416–431, 2011.
15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
16 Ariel Kulik, Kanthi K. Sarpatwar, Baruch Schieber, and Hadas Shachnai. Generalized Assignment via Submodular Optimization with Reserved Capacity. *CoRR*, abs/1907.01745, 2019. arXiv:1907.01745.
17 Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.
18 Hui Lin and Jeff Bilmes. Multi-document Summarization via Budgeted Maximization of Submodular Functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT'10, 2010.
19 G. L. Nemhauser and L. A. Wolsey. Best Algorithms for Approximating the Maximum of a Submodular Set Function. *Math. Oper. Res.*, 3(3):177–188, 1978.
20 Kanthi K. Sarpatwar, Baruch Schieber, and Hadas Shachnai. Generalized Assignment of Time-Sensitive Item Groups. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 24:1–24:18, 2018.
21 David B. Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62:461–474, 1993.
22 Maxim Sviridenko. A Note on Maximizing a Submodular Set Function Subject to Knapsack Constraint. *Operations Research Letters*, 32:41–43, 2004.
23 Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *40th Annual ACM Symposium on Theory of Computing, STOC, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 67–74, 2008.