The Ground-Set-Cost Budgeted Maximum Coverage Problem

Irving van Heuven van Staereling[1], Bart de Keijzer[1], and Guido Schäfer[1,2]

1 **Centrum Wiskunde & Informatica (CWI), Networks and Optimization Group, Amsterdam, The Netherlands.**
Email: heuven@cwi.nl, keijzer@cwi.nl, schaefer@cwi.nl

2 **Vrije Universiteit Amsterdam, Department of Econometrics and Operations Research, Amsterdam, The Netherlands**

Abstract

We study the following natural variant of the budgeted maximum coverage problem: We are given a budget B and a hypergraph $G = (V, E)$, where each vertex has a non-negative cost and a non-negative profit. The goal is to select a set of hyperedges $T \subseteq E$ such that the total cost of the vertices covered by T is at most B and the total profit of all covered vertices is maximized. Besides being a natural generalization of the well-studied maximum coverage problem, our motivation for investigating this problem originates from its application in the context of bid optimization in sponsored search auctions, such as Google AdWords.

It is easily seen that this problem is strictly harder than budgeted max coverage, which means that the problem is $(1 - 1/e)$-inapproximable. The difference of our problem to the budgeted maximum coverage problem is that the costs are associated with the covered vertices instead of the selected hyperedges. As it turns out, this difference refutes the applicability of standard greedy approaches which are used to obtain constant factor approximation algorithms for several other variants of the maximum coverage problem. Our main results are as follows:

- We obtain a $(1 - 1/\sqrt{e})/2$-approximation algorithm for graphs.
- We derive a fully polynomial-time approximation scheme (FPTAS) if the incidence graph of the hypergraph is a forest (i.e., the hypergraph is *Berge-acyclic*). We also extend this result to incidence graphs with a fixed-size feedback hyperedge node set.
- We give a $(1 - \varepsilon)/(2d^2)$-approximation algorithm for every $\varepsilon > 0$, where d is the maximum degree of a vertex in the hypergraph.

1998 ACM Subject Classification F.2.2 – Nonnumerical Algorithms and Problems

Keywords and phrases maximum coverage problem, approximation algorithms, hypergraphs, submodular optimization, sponsored search

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.51

1 Introduction

In the *budgeted maximum coverage problem* we are given a hypergraph $G = (V, E)$ with a non-negative cost $c(e) \in \mathbb{R}_{\geq 0}$ for every hyperedge $e \in E$ and a non-negative profit $p(i) \in \mathbb{R}_{\geq 0}$ for every vertex $i \in V$, and a non-negative budget $B \in \mathbb{R}_{\geq 0}$. The goal is to select a set of hyperedges $T \subseteq E$ whose total cost is at most B such that the total profit of all vertices covered by the hyperedges in T is maximized.

This is a fundamental combinatorial optimization problem with many applications in resource allocation, job scheduling and facility location (see, e.g., [7] for examples). Feige [5]

showed that this problem is not polynomial-time approximable within a factor of $(1 - 1/e)$ unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$, even if all hyperedges have unit cost. Khuller, Moss and Naor [10] derived a $(1 - 1/e)$-approximation algorithm for the budgeted maximum coverage problem (which is the best possible). Their algorithms are based on a natural greedy approach in combination with a standard enumeration technique. Similar approaches were used to derive constant factor approximation algorithms for several other variants and generalizations of the maximum coverage problem.

In this paper, we study the following natural variant of the budgeted maximum coverage problem, which we call the *ground-set-cost budgeted maximum coverage problem (GBMC)*: We are given a hypergraph $G = (V, E)$ with a non-negative cost $c(i) \in \mathbb{R}_{\geq 0}$ and a non-negative profit $p(i) \in \mathbb{R}_{\geq 0}$ for every vertex $i \in V$, and a non-negative budget $B \in \mathbb{R}_{\geq 0}$. For a subset $T \subseteq E$, define $c(T) = \sum_{i \in \cup T} c(i)$ and $p(T) = \sum_{i \in \cup T} p(i)$ as the total cost and profit, respectively, of all vertices covered by the hyperedges in T.[1] Our goal is to select a set of hyperedges $T \subseteq E$ such that the total cost $c(T)$ of all covered vertices is at most B and the total profit $p(T)$ of all covered vertices is maximized. To the best of our knowledge, this problem has not been studied before.

Note that a crucial difference here is that in our problem costs are incurred per covered vertex, while in the budgeted maximum coverage problem costs are incurred per selected hyperedge. Albeit seemingly minor, this change makes the problem much harder to tackle algorithmically. More specifically, most greedy approaches (which give rise to constant factor approximation guarantees for several variants of the maximum coverage problem) turn out to be inapplicable in our setting because of the following reason: The basic idea underlying these greedy approaches is to select in each iteration a hyperedge that is most *cost-efficient*, i.e., maximizes the ratio of the profit of newly covered vertices over the cost of selecting the hyperedge. A property that is crucially exploited in the analysis of these algorithms is that the cost for selecting a hyperedge is constant, i.e., its cost-efficiency can only decrease throughout the course of the algorithm (as more of its vertices get covered). However, this monotonicity property is no longer guaranteed in our setting because the cost for picking a hyperedge depends on the set of already covered vertices. In fact, it is not hard to see that the cost-efficiency of a hyperedge can change arbitrarily from one iteration to the next.

Our motivation for investigating the vertex-cost budgeted maximum coverage problem is two-fold: (i) It is a generalization of the well-studied maximum coverage problem and a natural variant of the budgeted maximum coverage problem. (ii) It is a fundamental combinatorial optimization problem having several applications in practice. Of particular importance is its relation to the problem of computing optimal bids in sponsored search auctions such as Google AdWords (details will be given in the full version of the paper).

Our contributions

The contributions presented in this paper are as follows:

1. We obtain a $(1 - 1/\sqrt{e})/2$-approximation algorithm for graphs (Sections 2 and 3).
   The main idea here is to reduce this problem to the budgeted maximum coverage problem with an exponential number of hyperedges. However, we do not need to generate the exponentially large instance explicitly; but instead we make use of a concise representation of the instance and show that such instances can be approximated in polynomial time,

[1] Throughout this paper, for a collection of sets F we write $\cup F$ to refer to the set $\cup_{S \in F} S$.

given that we have access to an oracle that can select in polynomial time a hyperedge with approximately highest profit per unit of cost. As a last step in our reduction, we prove that such an oracle exists.

2. We derive in Section 5 a pseudo-polynomial time algorithm for the case when the incidence graph of the hypergraph is a forest (i.e., the hypergraph is *Berge-acyclic*). Further, we adapt this algorithm into a fully polynomial-time approximation scheme (FPTAS).

   At the core of this algorithm lies a bi-level dynamic program. The case of forests is important in its own right and, additionally, this algorithm constitutes an important building block of our $O(1/d^2)$-approximation algorithm (see Contribution 4).

3. In Section 6, we extend the above algorithm to a pseudo-polynomial time algorithm for incidence graphs with a bounded set of nodes that covers all cycles (i.e., the general case, but parametrized).

   More specifically, we show that for any incidence graph with a fixed-size *feedback hyperedge node set*, i.e., a hyperedge node set such that removing it from the incidence graph leaves no cycles, there exists a pseudo-polynomial time algorithm for the GBMC problem.

4. We give a $(1-\varepsilon)/(2d^2)$-approximation algorithm for every $\varepsilon > 0$ for the general case, where d is the maximum degree of a vertex in the hypergraph (Section 4).

   In this algorithm, we first decompose the incidence graph of the hypergraph into a collection of at most d trees for which we compute an approximate solution by using our FPTAS for forests above. From this we then extract a solution that is feasible for the original instance and guarantees an approximation ratio of at least $(1-\varepsilon)/(2d^2)$.

Related work

Much literature is available on the maximum coverage problem and its variants (see, e.g., [2, 4, 10] and the references therein). Most related to our problem is the budgeted maximum coverage problem [10]. As outlined above, the greedy approach of [10] cannot take into account that the costs are incurred per vertex instead of per set. Moreover, in [4], a generalized version of the budgeted maximum coverage problem is studied, but this generalization does not include GBMC as a special case.

Note that our GBMC problem on graphs reduces to the knapsack problem if the incidence graph is a matching. This problem is known to be weakly NP-hard and admits an FPTAS (see, e.g., [9]).

Our GBMC problem is related to the *budgeted bid optimization problem*. This problem was first proposed in the paper by Feldman et al. [6]. The authors derive a $(1-1/e)$-approximation algorithm if the budget constraint is *soft*, i.e., has to be met in expectation only. In contrast, in the budgeted bid optimization problem considered here, this budget constraint is hard.

The GBMC problem can be seen as a special case of a more general set of problems where we have to maximize a submodular profit function subject to the constraint that a submodular cost function does not exceed a given budget. This can be seen by considering the set of hyperedges to be the ground set of the submodular functions. However, when we have oracle access to both submodular functions, it has been shown that this more general problem is not approximable within a factor of $\log(m)/\sqrt{m}$, where m is the number of elements in the ground set. This holds even for the special case that the objective function is the modular function that returns the cardinality of the set. This follows from Theorem 4.2 in [12]; see also [8].

Preliminaries

For an integer $a \in \mathbb{N}$, we write $[a]$ and $[a]_0$ to denote the sets $\{1, \ldots, a\}$ and $\{0, 1, \ldots, a\}$ respectively. When F is a family of sets, we write $\bigcup F$ to denote the set $\bigcup_{S \in F} S$.

Let $G = (V, E)$ be a hypergraph. The *incidence graph* $I(G)$ of G is defined as the bipartite graph $I(G) = (E \cup V, H)$ with $H = \{\{e, v\} \mid v \in e\}$. We say that G is *acyclic* if its incidence graph $I(G)$ does not contain a cycle. Given a subset $E' \subseteq E$, we use $G[E']$ to refer to the *subgraph* of G induced by the hyperedges in E', i.e., $G[E'] = (V', E')$ with $V' = \cup E'$. A hypergraph T is called a *subtree* of G if T is a subgraph of G that is acyclic.

Throughout this paper we will use the convention that when discussing a hypergraph, n denotes the number of vertices of the hypergraph and m denotes the number of hyperedges of the hypergraph. Moreover, in the remainder of this paper, we assume without loss of generality that all costs (on the nodes or edges) are strictly positive.

It is not hard to prove that GBMC cannot be approximated to within a factor of $(1-1/e)$ in polynomial time, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$ (details will be provided in the full version of the paper).

Due to space limitations, some figures and technical content is omitted from this paper and will be provided in the full version.

2 Budgeted Maximum Coverage with Oracles

In this section we first consider the classical budgeted maximum coverage problem. The result presented in this section will serve as a building block in the approximation algorithm presented in the next section, for solving GBMC on graphs.

A polynomial-time $(1-1/e)$-approximation algorithm for the budgeted maximum coverage problem was previously given in [10]. In the same paper, various simpler algorithms with worse approximation factors are presented. In this section, we present a variation of one of these algorithms that achieves a $(1 - 1/e)/2$-approximation guarantee, which can run even if the algorithm is not granted direct access to the input instance. We make this precise in the following definition.

▶ **Definition 1** (cost-efficiency oracle)**.** Let $I = (G = (V, E), c, p, B)$ be an instance of the budgeted maximum coverage problem, i.e., $G = (V, E)$ is a hypergraph, $c : E \to \mathbb{Q}_{\geq 0}$ is a function that specifies a cost $c(e)$ for each hyperedge $e \in E$, $p : V \to \mathbb{Q}$ is a function that specifies a profit $p(i)$ for each vertex $i \in V$, with $B \in \mathbb{Q}$ the budget. For $\alpha \in [0, 1]$, an *α-approximate cost-efficiency oracle* for I is a function $f_I : 2^V \to E$ that maps a set of vertices $S \subseteq V$ to a hyperedge $e \in E$ such that $c(e) \leq B$ and

$$\sum_{i \in e \setminus S} \frac{p(i)}{c(e)} \geq \alpha \cdot \sum_{i \in e' \setminus S} \frac{p(i)}{c(e')}.$$

for all $e' \in E$ with $c(e') \leq B$. Thus, a cost-efficiency oracle takes as input vertex set S and selects the hyperedge with the approximately highest cost-efficiency (up to a factor α), excluding the profit that would be contributed by vertices in S. Only hyperedges of which the cost does not exceed the budget are considered.

Let $I = (G = (V, E), c, p, B)$ be an instance of the budgeted maximum coverage problem, and let f_I be an α-approximate cost-efficiency oracle for this instance for some $\alpha \in (0, 1]$. Consider now the following greedy algorithm $\mathcal{A}$ that takes as input only the cost-efficiency oracle f_I.

1. Set $S := \varnothing$ and $X := \varnothing$. Throughout the execution of the algorithm, X represents a feasible solution and S represents the set of vertices covered by X.
2. Let $e := f_I(S)$. If $S = V$ (i.e., there is no profitable hyperedge left) or if $c(e) + \sum_{e' \in X} c(e') > B$ (i.e., adding the hyperedge to X would exceed the budget), go to Step 3. Otherwise, set $X := X \cup \{e\}$, set $S = \bigcup X$, and repeat this step.
3. Output the solution with the highest total profit among the two solutions X and $\{e\}$.

▶ **Theorem 2.** *Algorithm $\mathcal{A}$ outputs an $(1 - 1/e^{\alpha})/2$-approximate solution to I in time $O(n \cdot t)$, where t is the amount of time it takes to evaluate f_I.*

The approximation factor is obtained by following rather closely the analysis given in [10] for a similar algorithm (that works without oracle access).

3 GBMC on Graphs

In this section, we present a $(1 - 1/\sqrt{e})/2$-approximation algorithm for the GBMC problem when the hypergraph is a graph. We do this by reducing the problem to the budgeted maximum coverage problem. An instance I of GBMC is reduced to an instance $r(I)$ of budgeted maximum coverage on the same set of vertices, such that the optimal solution of $r(I)$ has the same profit as the optimal solution of I. The instance $r(I)$ may have a superpolynomial number of hyperedges. However, instead of generating the budgeted maximum coverage instance explicitly, we construct only a 1/2-approximate cost-efficiency oracle $f_{r(I)}$ for $r(I)$. We then use Algorithm $\mathcal{A}$ on $f_{r(I)}$ in order to obtain a $(1 - 1/\sqrt{e})/2$-approximately optimal solution to $r(I)$ in polynomial time. Last, we show how to transform in polynomial time a feasible solution for $r(I)$ into a feasible solution for I with equal profit.

We begin by defining our reduction r.

▶ **Definition 3.** Let $I = (G = (V, E), c, p, B)$ be an instance of GBMC where G is a graph. Define the budgeted maximum coverage instance $r(I)$ as $r(I) = (G' = (V, E'), c', p, B)$, where

$$E' = \bigcup_{i \in V} E'_i \qquad \text{and} \qquad E'_i = \{S \cup \{i\} \mid \forall i' \in S : \{i', i\} \in E\},$$

that is, E'_i consists of the hyperedges X such that i is in X and all other vertices in X are connected to i by an edge. In other words, E' are all hyperedges corresponding to the stars of G. The cost function c' assigns a cost to each *hyperedge*: for a hyperedge $e \in E'$ we set $c'(e) = \sum_{i \in e} c(i)$. Note that c is a function that assigns a cost to each *vertex*, while c' is a function that assigns a cost to each *hyperedge in* E'. Note that the vertex sets, profit functions, and budgets of I and $r(I)$ are equal.

We first show that every feasible solution X' for $r(I)$ can be transformed into a feasible solution X for I in polynomial time such that the profit is preserved. Consider the following function g_I that maps solutions of $r(I)$ to I:

▶ **Definition 4.** Let $I = (G = (V, E), c, p, B)$ be an instance of GBMC and let X' be a feasible solution for $r(I) = (G' = (V, E'), c', p, B)$. The function g_I maps X' to the following solution for I.

$$g_I(X') = \left\{ \{i', i\} \in E \,\middle|\, \{i', i\} \in \bigcup X \right\}.$$

In words, $g_I(X')$ is the set of edges of G that are contained in a hyperedge of X'.

▶ **Lemma 5.** *Let X' be a feasible solution for $r(I)$. The edge set $g_I(X')$ is computable in time $O(mn|X'|)$. Moreover, the solution $g_I(X')$ is feasible (i.e., the total cost of all vertices covered by $g_I(X')$ does not exceed B). Also, $p(X') = p(g_I(X'))$.*

Proof. For the first claim, observe that for each hyperedge in X' and edge in E we need to check if that edge is contained in the hyperedge. This can be done in $O(n)$ time.

The second claim follows from the fact that the edge set $g_I(X')$ covers the same vertex set as X', and by definition

$$B \geq \sum_{e \in X'} c'(e) = \sum_{i \in \bigcup X'} c(i) \cdot |\{e \in X' : i \in e\}| \geq \sum_{i \in \bigcup X'} c(i) = \sum_{i \in \bigcup X'} c(i).$$

The third claim follows from the fact that the edge set $g_I(X')$ covers the same vertex set as X'. ◀

Next we show that the optimal solution for I is at most the profit of the optimal solution for $r(I)$. (Combined with the previous lemma, this entails that the optimal profits of I and $r(I)$ are equal.)

▶ **Lemma 6.** *Let p_{opt} be the maximum profit achievable in instance I. There exists a solution for $r(I)$ with profit p_{opt}.*

Proof. Let X be a profit-maximizing feasible solution for I. Assume without loss of generality that all paths in X are of size at most 2. In other words: no edge in X covers two vertices that are both covered by another edge (such an edge can be removed from X without decreasing the profit). Under this assumption, X is a set of stars. We construct from X a feasible solution X' for $r(I)$ that has the same profit, as follows. We define X' to be the collection of hyperedges that correspond to the maximal stars of X, i.e., for each maximal star of X, we add to X' the hyperedge consisting of the vertices covered by the star.

Since no pair of hyperedges in X' intersects, by definition of c' the total cost $\sum_{e \in X'} c'(e)$ equals $\sum_{i \in \bigcup X} c(i) < B$, and therefore X' is a feasible solution for $r(I)$. Moreover, X' and X cover the same set of vertices, and therefore profits of X in I equals the profit of X' in $r(I)$. ◀

A final ingredient that we need is a 1/2-approximate cost-efficiency oracle f for $r(I)$.

▶ **Definition 7.** We define the function f algorithmically as follows. Let S be the input argument to f. (As a reminder, S represents the set of vertices already covered during the execution of algorithm $\mathcal{A}$.) The high level idea is that we compute for each vertex i a set of vertices e_i in the star centered at i. Our goal for each of these stars is to select for each such i the substar with the (approximately) highest possible cost-efficiency, such that the cost of the vertices in the substar does not exceed the budget. We output the set in $\{e_i : i \in V\}$ that has the highest cost-efficiency.

1. Let V' be subset of vertices of V that have at least one neighbor not in S. For each $i \in V'$ (note that i itself may be in S):
   - **a.** Initialize $e_i := \{i\}$, and $d_i = c(i)$. If $i \in S$, set $n_i := 0$, and otherwise set $n_i := p(i)$.
   - **b.** Order non-increasingly the vertices i' that are not in S and are attached to i in graph G, according to ratio $p(i')/c(i')$. Denote this ordering by σ_i.
   - **c.** Let i' be the next vertex of σ_i (starting with the first vertex). If $(n_i+p(i'))/(d_i+c(i')) \geq n_i/d_i$, then add i' to e_i, set $n_i := n_i+p(i')$, and set $d_i := d_i+c(i')$, and repeat this step in case the total cost of e_i does not exceed B. Otherwise, if $(n_i+p(i'))/(d_i+c(i')) < n_i/d_i$ or if e_i exceeds the budget, stop iterating this step.

d. If the total cost of e_i lies within the budget, skip this step. Otherwise, let i' be the vertex last added in the previous step (i.e., the vertex in e_i with the least $p(i')/c(i')$). We consider two substars of e_i that are within the budget: The one consisting only of vertices i and i', and the one consisting of vertices $e_i \setminus \{i'\}$. We set e_i to be the substar with the highest cost-efficiency. Formally:
   i. If $i \notin S$: if $(p(i)+p(i'))/(c(i)+c(i')) \geq (n_i - p(i'))/(d_i - p(i'))$ then set $e_i = \{i, i'\}$, $n_i := p(i)+p(i')$, and $d_i := c(i)+c(i')$. Otherwise set $e_i := e_i \setminus \{i'\}$, $n_i := n_i - p(i')$, and $d_i := d_i - c(i')$.
   ii. If $i \in S$: if $p(i')/(c(i)+c(i')) \geq (n_i - p(i'))/(d_i - p(i'))$ then set $e_i := \{i, i'\}$, $n_i := p(i')$, and $d_i := c(i)+c(i')$ otherwise set $e_i := e_i \setminus \{i'\}$, $n_i := n_i - p(i')$, and $d_i := d_i - c(i')$.

2. Output the set in $\{e_i : |e_i| \geq 2 \wedge i \in V'\}$ with the highest cost-efficiency (i.e., the ratio n_i/d_i).

▶ **Lemma 8.** *The function f is a $1/2$-approximate cost-efficiency oracle for $r(I)$ and can be computed in time $O(n^2)$.*

Proof. It is easy to see that the set output by f is always a hyperedge in E', as it only outputs sets of hyperedges that correspond to stars of G. Moreover, in the last step, it is easy to verify that the set $\{e_i : |e_i| \geq 2 \wedge i \in V'\}$ is never empty when $S \neq V$. This implies that f is a valid cost-efficiency oracle. From the description of the algorithm above, it is also straightforward to see that f runs in time n^2: For each vertex, all neighbors are considered, where processing each neighbor takes a constant amount of time. (Not taking into account the bit-complexity of the arithmetic operations in this analysis, although the runtime would remain polynomial if we would take this aspect into account.)

What still needs to be proved is the approximation factor. Let $e_1, e_2, \ldots$ be the sets used in Step 2 of the algorithm. It suffices to show that for each $i \in V'$ for which it holds that $|e_i| \geq 2$, the ratio n_i/d_i is at least $(1/2) \cdot \sum_{i \in e' \setminus S} p(i)/c(e')$ for all $e' \in E'_i$. In words, the cost-efficiency n_i/d_i of the set e_i is at least half the maximum cost-efficiency among all hyperedges in E'_i (with respect to the input set S). (Note that we need not consider those $i \in V'$ for which $|e_i| = 1$: It can be easily verified that in this case, the optimal star centered at i is a single edge $\{i, i'\}$. This edge is also in $E'_{i'}$, and it is necessarily true that $|e'_i| \geq 2$.)

Let $i \in V'$ such that $|e_i| \geq 2$. Denote by $\Gamma(i)$ the vertices attached to i that are not in S. We will compare $|e_i|$ to an optimal *fractional* solution x: In this fractional solution each of the vertices i' attached to i (and not in S) is picked with a certain fraction $x_{i'} \in [0, 1]$, and vertex i is selected with fraction $x_i = 1$. The cost-efficiency is defined as $\frac{p(i)+\sum_{i' \in \Gamma(S)} x_{i'} p(i')}{\sum_{i' \in \Gamma(S)} x'_i c(i')}$ if $i \notin S$, and otherwise as $\frac{\sum_{i' \in \Gamma(S)} x_{i'} p(i')}{\sum_{i' \in \Gamma(S)} x'_i c(i')}$. Then it holds that the cost-efficiency of e_i^{frac} exceeds the cost-efficiency of the hyperedge $e_i^* \in E'_i$ that maximizes $\sum_{i \in e^* \setminus S} p(i)/c(e^*)$, which would be the optimal integral solution.

We claim that x is obtained by greedily selecting vertices in $\Gamma(i)$ according to non-increasing cost-efficiency (i.e., according to the order σ_i as given in Definition 7). A considered vertex is selected with the highest possible fraction as long as the budget is not exceeded, and as long as adding the vertex increases the cost-efficiency of the solution. Hence, in x all vertices of $\Gamma(i)$ are selected with either fraction 0 or 1, except at most one vertex, which is selected with a fraction in $(0, 1)$.

To see why this is true, suppose for contradiction that x has a different structure. In that case, if there is a vertex $i' \in \Gamma(i)$ with $x_{i'} > 0$ such that the cost-efficiency of i' is less

than the cost-efficiency of x, then setting $x_{i'}$ to 0 will increase the cost-efficiency of the solution. Therefore, we may assume that the only vertices that are selected with a positive fraction, are vertices that have a cost-efficiency of at least the cost-efficiency of x. We can then consider the following operation: There must be two vertices $i', i'' \in \Gamma(i)$ for which it holds that $x'_i < 1$, $x''_i > 0$, and the cost-efficiency of i' exceeds that of i''. In that case, decreasing $x_{i''}$ by an amount ϵ and increasing $x_{i'}$ by a maximal amount would increase the cost-efficiency (for a suitably small choice of ϵ), which is a contradiction to x being optimal. This shows that x is obtained by the aforementioned greedy procedure.

Next, we observe that if x happens to be integral, then the set of integrally selected vertices is precisely e_i, which means that e_i is the vertex set that maximizes the cost-efficiency. In this case the claim is proved. We now consider the case that x is not integral. From now on, let i' be the vertex that is fractionally selected in x and let S' be the integral vertices of x excluding i, i.e., $S' = \{i'' : i'' \neq i \wedge x_{i''} = 1\}$. It follows from Definition 7 that e_i is either the set $\{i\} \cup S$ or the set $\{i, i'\}$.

We distinguish four (very similar) subcases.

- We first consider the case that $i \notin S$ and $p(i') \geq \sum_{i'' \in S'} p(i'')$. Because x is the optimal fractional solution, the cost-efficiency of $S' \cup i, i'$ exceeds the optimal fractional solution and thus also the cost-efficiency of the optimal hyperedge e^*_i. Therefore, we conclude that the cost-efficiency of e_i is at least

$$\frac{p(i) + p(i')}{c(i) + c(i')} \geq \frac{p(i) + p(i')}{c(i) + c(i') + \sum_{i'' \in S'} c(i'')} \geq \frac{1}{2} \cdot \frac{p(i) + p(i') + \sum_{i'' \in S'} c(i'')}{c(i) + c(i') + \sum_{i'' \in S'} c(i'')} \geq \frac{1}{2} \cdot \frac{\sum_{i'' \in e^*_i} p(i)}{\sum_{i'' \in e^*_i} c(i)},$$

  as needed.

- In case $i \notin S$ and $p(i') < \sum_{i'' \in S'} p(i'')$ we similarly obtain that the cost-efficiency of e_i is at least

$$\frac{p(i) + \sum_{i' \in S'} p(i'')}{c(i) + \sum_{i'' \in S'} c(i'')} \geq \frac{p(i) + \sum_{i'' \in S'} p(i'')}{c(i) + c(i') + \sum_{i'' \in S'} c(i'')} \geq \frac{1}{2} \cdot \frac{p(i) + p(i') + \sum_{i'' \in S'} c(i'')}{c(i) + c(i') + \sum_{i'' \in S'} c(i'')} \geq \frac{1}{2} \cdot \frac{\sum_{i'' \in e^*_i} p(i)}{\sum_{i'' \in e^*_i} c(i)}.$$

- The remaining two cases are analogous to the above two, where we replace $p(i)$ with 0.

◀

We are now ready to present the algorithm for GBMC on graphs, which we refer to as Algorithm $\mathcal{B}$. The algorithm is defined as follows. Let $I = (G = (V, E), c, p, B)$ be an input instance of GBMC where G is a graph.

- Run algorithm $\mathcal{A}$ on the 1/2-approximate cost-efficiency oracle f of Definition 7. This results in a solution X' for instance $r(I)$ (where $r(I)$ is given in Definition 3).
- Compute and output $g_I(X')$ (see Definition 4).

The correctness, polynomial runtime, and approximation factor of $(1 - 1/\sqrt{e})/2$ of algorithm $\mathcal{B}$ follow directly from the lemmas and definitions above. Note that the bound on the runtime can most likely be improved by a more careful analysis, but that is beyond the scope and goal of this work.

4 GBMC with bounded degree vertices

In this section we derive an approximation algorithm for GBMC for arbitrary hypergraphs $G = (V, E)$ with approximation ratio of $\mathcal{O}(1/d^2)$, where d refers to the maximum *frequency* of a node in a decomposition of G into trees. We first define formally the notions of *trees of hypergraphs*, and *frequency*.

A *tree of a hypergraph* G is defined as a partial hypergraph (i.e., a hypergraph that can be obtained by removing from each hyperedge a set of vertices) of which the incidence graph is a tree (i.e., a partial hypergraph that is *Berge-acyclic*). A *decomposition of a hypergraph* G *into trees* is a collection of trees of G such that (i) the incidence graphs of any two trees in the collection have disjoint edge sets, and (ii) the union of the incidence graphs of these trees equals the incidence graph of G. For a node $i \in V$, define the *frequency* d_i of i with respect to the tree collection $\mathcal{T}$ as the number of times i occurs in a tree of the collection, i.e., $d_i = |\{T_t \in \mathcal{T} \mid i \in V_t\}|$. Let the *maximum frequency* of a tree collection $\mathcal{T}$ be defined as $d = \max_{i \in V} d_i$. Therefore, in the worst case, d is the maximum degree of a node, as we can always decompose a hypergraph into subtrees that each consist of a single hyperedge of G. Our algorithm, which we name Algorithm $\mathcal{C}$, proceeds in three steps:

Step 1: Decomposition into trees. Let $I = (G = (V, E), p, c, B)$ be an instance of GBMC. We first decompose G into a collection of subtrees of G as follows: Initialize $t = 0$ and let $G_0 = G$ be the initial hypergraph. For $t \geq 0$ extract a subtree $T_{t+1} = (V_{t+1}, E_{t+1})$ from G_t and let $G_{t+1} = G_t \setminus T_{t+1}$ be the hypergraph that remains if we remove all hyperedges in E_{t+1} (but not the nodes) from G_t. Repeat the above procedure until eventually we obtain a graph G_z whose set of hyperedges is empty. Let $\mathcal{T} = \{T_1, \ldots, T_z\}$ be the collection of subtrees extracted throughout this procedure. Note that by construction, for every two distinct trees $T_t, T_{t'} \in \mathcal{T}$ the set of hyperedges E_t and $E_{t'}$ are disjoint.

Using the above decomposition, we now define a new instance $I' = (G', p', c', B')$ of GBMC. The hypergraph G' consists of all trees $T_1, \ldots, T_z$, where each tree $T_t = (V_t, E_t)$, $t \in [z]$, has its own "representative" for each node in V_t (with costs and profits being identical to the original ones). Thus, each node $i \in V$ has at most d_i representatives in G' and all trees in G' are node-disjoint. Finally, the budget B' of I' is set to $B' = dB$.

Step 2: Bin-packing the optimal solution of the decomposed instance. We compute a $(1 - \varepsilon)$-approximate solution X' for I' which respects the overall budget $B' = dB$ and also ensures that the total cost of every $T_t \in \mathcal{T}$ is at most B. The latter condition can easily be incorporated in our dynamic program for forests (thus also in our FPTAS) in Section 5.

The next step is to partition the trees in $\mathcal{T} = \{T_1, \ldots, T_z\}$ into at most $2d$ sets $\mathcal{T}_1, \ldots, \mathcal{T}_{2d}$ such that the total cost (according to X') in each set does not exceed B. This is in essence a bin-packing problem (i.e., packing a set of items of varying weights in a set of bins of limited capacity). Because every tree induces cost at most B and the total cost is at most dB, standard bin-packing arguments show that such a partition exists and can be computed in polynomial time [13].[2]

Step 3: Obtaining a solution for the original instance. Let $\mathcal{T}$ be a set of maximum profit (according to X') among the sets $\mathcal{T}_1, \ldots, \mathcal{T}_{2d}$. We obtain the solution X from X' by

[2] To clarify: We can view each tree as an item of weight equal to the cost induced by X'. The goal then is to pack these items into bins of capacity B.

picking all hyperedges chosen in $\mathcal{T}$. Note that X is a feasible solution for I' but also for I as argued in Step 2. The algorithm outputs X.

▶ **Theorem 9.** *Algorithm $\mathcal{C}$ is a $(1-\varepsilon)/(2d^2)$-approximation algorithm for GBMC that runs in polynomial time, where d is the maximum frequency of a node.*

Proof. It is clear that the algorithm runs in polynomial time. Moreover, the algorithm outputs a feasible solution because the total cost induced by the nodes covered by X' in $\mathcal{T}$ is at most B (by construction). Therefore, the total cost of X in I is at most B.

It remains to analyze the approximation ratio. Let OPT_I be the optimal profit of the original instance I and let $OPT_{I'}$ be the optimal profit of the decomposed instance. Note that any feasible solution for I is also feasible for I'. This follows because the total cost of a solution for I is at most d times larger in I' and $B' = dB$. Therefore, the total profit $p_{I'}(X')$ of X' in I' is at least $(1-\varepsilon)OPT_{I'} \geq (1-\varepsilon)OPT_I$.

Because we choose the maximum set $\mathcal{T}$ among the $2d$ many sets, the total profit of X in I' is at least $(1-\varepsilon)OPT_I/(2d)$. Also, the total profit of X in I is at most a factor d less than the total profit it induces in I'. Therefore, the total profit $p_I(X)$ of X in I satisfies $p_I(X) \geq (1-\varepsilon)OPT_I/(2d^2)$, which completes the proof. ◀

5 GBMC when the incidence graph is a forest

In this section, we derive a bi-level dynamic program for the case when the incidence graph is a forest. We refer to this special case as GBMC-Forest. We also show that our dynamic program can be turned into an FPTAS, for which we introduce P as the maximum profit of a vertex, i.e., $P = \max_{v \in V} p(v)$.

▶ **Theorem 10.** GBMC-Forest *can be solved optimally in time $\mathcal{O}(mn^3P^2)$.*

Suppose the given incidence graph is a forest and consists of z trees $T_1, \ldots, T_z$. In order to facilitate the exposition of our dynamic program we combine these trees simply into a single tree T as follows. Introduce an artificial hyperedge e_0, representing the root of the tree. Furthermore, we introduce for each tree T_t with $t \in [z]$ a dummy vertex node d_t with zero profit and cost, i.e., $p(d_t) = c(d_t) = 0$, and connect it to e_0. Finally, we connect d_t to its respective tree T_t by adding an edge in the incidence graph from d_t to an arbitrary hyperedge from T_t.

This yields a bipartite graph that is a single tree. Note that the nodes along a path from the root to any other node are alternately hyperedge/vertex nodes. Assume we "unfold" this tree in order to draw the bipartite graph in a layered manner, as illustrated in Figure 1.

Our dynamic program processes the unfolded tree T in a bottom-up manner. It consists of two separate dynamic programs, one for the hyperedges, and one for the vertices. We describe these programs informally in this section (technical details of the dynamic program will be given in the full version).

Consider an arbitrary subtree in T rooted at either a node represented by a hyperedge or vertex. Both dynamic programs rely on the fact that a subset of this subtree can be solved to optimality, which immediately can be used to solve a greater subset to optimality. In case the subtree is rooted at a node represented by a hyperedge, we consider the subtree up until the first s children in the subtree of the hyperedge. Once the optimal solutions (minimum required cost to obtain a specific profit, if possible) are known for every possible profit (upper bounded by nP), it is possible to find optimal solutions for the subtree until the first $s+1$ children by linear enumeration. A similar, but slightly adapted method works in case the subtree is rooted by a vertex rather than a hyperedge.

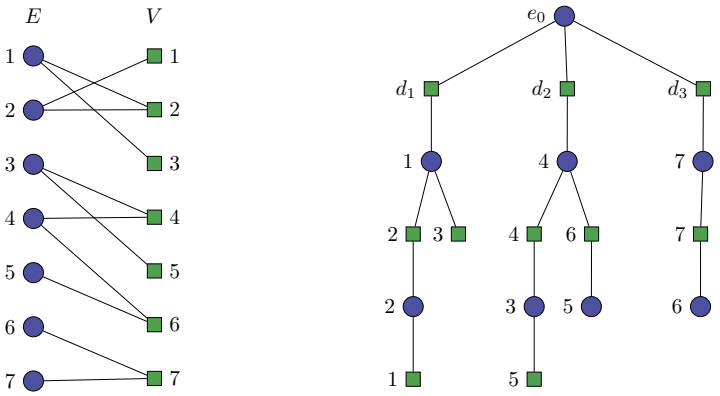

Figure 1 Example of a incidence graph (left) and its "unfolded' tree (right).

Furthermore, we can employ standard techniques (profit truncation) to turn the above pseudo-polynomial time algorithm into an FPTAS, i.e., an algorithm that takes an error parameter $\varepsilon > 0$ and computes in time polynomial in the input size and $1/\varepsilon$ a $(1-\varepsilon)$-approximation to the optimal solution.

▶ **Theorem 11.** *There exists an FPTAS for* GBMC-Forest *that runs in time* $\mathcal{O}(mn^5/\epsilon^2)$.

6 GBMC with a bounded size feedback vertex set

We have shown in the previous section that GBMC-Forest can be solved in pseudo-polynomial time and that there is an FPTAS. This implies that the inapproximability of the general problem is caused by the cycles in the incidence graph. In this section, we provide a fixed parameter tractability result that allows us to handle incidence graphs with cycles. The fixed parameter is the minimum number α of hyperedge nodes that we need to remove in order to make the incidence graph acyclic.

To provide an initial intuition, construct a graph $G' = (E, E')$ where every hyperedge $e \in E$ is represented by a node, and two nodes are connected if and only if the corresponding hyperedges share at least one element. This defines the edge set E' in the new graph. Consider the case in which G contains solely one cycle. Select a hyperedge node of the cycle and fix whether this hyperedge is chosen or not in a solution (i.e., set $x = 0$ or 1). We then consider the reduced problem in which hyperedge e is removed. The incidence graph of the reduced problem is a forest and can thus be solved optimally by using the pseudo-polynomial time algorithm (or approximately by using the FPTAS) of the previous section. Solving this GBMC-Forest problem for every possible choice of hyperedge to remove, and taking the best solution, yields a solution to the general GBMC problem for the instance with one cycle. Thus, if the graph contains one cycle, the running time is multiplied by 2. We can extend this idea to more general graphs.

Define α as the minimum number of hyperedges whose removal turn the graph into a forest, i.e., α is the cardinality of the *minimum feedback vertex subset* of the hyperedge nodes of the incidence graph. We refer to the latter as the *minimum feedback hyperedge node set*. Then the running time of the algorithms mentioned in the previous sections is multiplied by a factor of 2^α, because it is necessary to solve the problem on a forest for every combination on those α hyperedges.

The problem of finding a minimum feedback vertex set is NP-hard in general, but it is fixed parameter tractable. Cao et al. [3] give an $\mathcal{O}(3.83^{\alpha}\alpha n^2)$ time algorithm to solve the problem, where n here refers to the number of nodes in the graph. We use this to prove the following theorem.

▶ **Theorem 12.** *GBMC is solvable in $\mathcal{O}(mn^3P^2 2^{\alpha} + 3.83^{\alpha}\alpha m^2)$ time, where α is the size of the minimum feedback hyperedge node set.*

All that needs to be shown is how to use the $\mathcal{O}(3.83^{\alpha}\alpha n^2)$ algorithm of [3] in order to find a minimum feedback vertex set restricted to only the hyperedge nodes of the incidence graph. This is straightforward: We reduce the incidence graph of G to the aforementioned multigraph $G' = (E, E')$ with only E as its vertex set. The edge set E' is constructed as follows: there exists an edge between two hyperedges if they share at least one vertex in the original graph. It is now easy to see that there is a one-to-one correspondence between the cycles in the incidence graph of G and the cycles in G', and each cycle in the incidence graph of G corresponds to a cycle in G' on the same set of vertices. Therefore, a minimum feedback vertex set of G' corresponds to a minimum feedback hyperedge node set of G, and the algorithm of Cao et al. will find such a set in $\mathcal{O}(3.83^{\alpha}\alpha n^2)$ time.

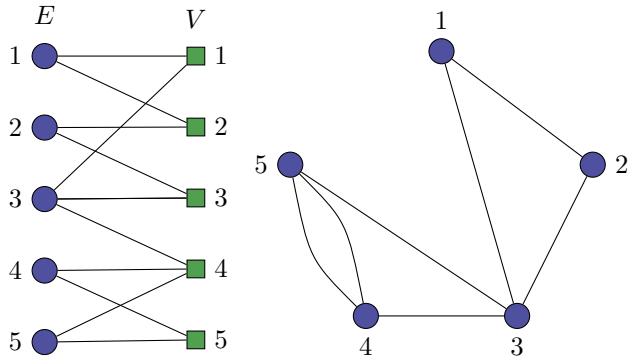

Figure 2 Example of a transformation to the feedback vertex set problem

The transformation is illustrated in Figure 2. There, hyperedge 3 and 5 are connected by vertex 4 in (the incidence graph representation of) G, thus an edge $\{3, 5\}$ is added in G'. The edge labels are omitted. Note that hyperedge 4 and 5 are connected by two edges, because they are both connected to vertex 4 and vertex 5.

7 Conclusions

In this paper we have presented various approximation algorithms for important special cases of the GBMC problem. Clearly, the most interesting open problem that remains to be solved is whether there exists a constant factor approximation algorithm for the general GBMC problem that runs in polynomial-time. Such a result would form a very interesting contrast with the inapproximability result for the problem of submodular function maximization with a submodular budget constraint under the oracle access model, which we mentioned in the introduction.

An interesting and challenging intermediate goal would be to find a constant factor approximation algorithm for the case that the hyperedges have a fixed size k. Algorithm $\mathcal{A}$ and Theorem 2 might serve as a useful tool for achieving this goal.

References

1 N. Archak, V. Mirrokni, and S. Muthukrishnan. Budget optimization for online campaigns with positive carryover effects. In *Proceedings of the 8th international conference on Internet and Network Economics*, pages 86–99. Springer-Verlag, 2012.

2 G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

3 Y. Cao, J. Chen, and Y. Liu. On feedback vertex set new measure and new structures. In *Proceedings of the 12th Scandinavian conference on Algorithm Theory*, pages 93–104. Springer-Verlag, 2010.

4 R. Cohen and L. Katzir. The generalized maximum coverage problem. *Information Processing Letters*, 108(1):15–22, 2008.

5 U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

6 J. Feldman, S. Muthukrishnan, M. Pál, and C. Stein. Budget optimization in search-based advertising auctions. In *Proceedings of the 8th ACM conference on electronic commerce*, pages 40–49, New York, NY, USA, 2007. ACM.

7 D.S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., 1997.

8 R.K. Iyer and J.A. Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *Advances in Neural Information Processing Systems*, pages 2436–2444. MIT Press, 2013.

9 H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer-Verlag, 2004.

10 S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

11 P. Maillé, E. Markakis, M. Naldi, G.D. Stamoulis, and B. Tuffin. Sponsored search auctions: an overview of research with emphasis on game theoretic aspects. *Electronic Commerce Research*, 12(3):265–300, 2012.

12 Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal of Computing*, 40(6):1715–1737, 2011.

13 V.V. Vazirani. *Approximation algorithms*. Springer-Verlag, 2003.

14 Y. Zhou and V. Naroditskiy. Algorithm for stochastic multiple-choice knapsack problem and application to keywords bidding. In *Proceedings of the 17th international conference on world wide web*, pages 1175–1176, New York, NY, USA, 2008. ACM.