Size-Constrained Weighted Set Cover

Lukasz Golab [1], Flip Korn [2*], Feng Li [3], Barna Saha [4*] and Divesh Srivastava [5]
[1]*University of Waterloo, Canada,* lgolab@uwaterloo.ca
[2]*Google Research,* flip@google.com
[3]*National University of Singapore,* li-feng@comp.nus.edu.sg
[4]*University of Massachusetts, Amherst,* barna.cs@gmail.com
[5]*AT&T Labs - Research, USA,* divesh@research.att.com
[*] *Work done while the authors were at AT&T Labs - Research*

***Abstract*—In this paper, we introduce a natural generalization of WEIGHTED SET COVER and MAXIMUM COVERAGE, called *Size-Constrained Weighted Set Cover*. The input is a collection of n elements, a collection of weighted sets over the elements, a size constraint k, and a minimum coverage fraction $\hat{s}$; the output is a sub-collection of up to k sets whose union contains at least $\hat{s}n$ elements and whose sum of weights is minimal. We prove the hardness of approximation of this problem, and we present efficient approximation algorithms with provable quality guarantees that are the best possible. In many applications, the elements are data records, and the set collection to choose from is derived from combinations (patterns) of attribute values. We provide optimization techniques for this special case. Finally, we experimentally demonstrate the effectiveness and efficiency of our solutions.**

I. INTRODUCTION

In this paper, we study a generalization of WEIGHTED SET COVER and MAXIMUM COVERAGE [10], called *Size-Constrained Weighted Set Cover*. The input is a collection of n elements, a collection of weighted sets over the elements, a size constraint k, and a minimum coverage fraction $\hat{s}$. The output is a sub-collection of up to k sets whose union contains at least $\hat{s}n$ elements and whose sum of weights is minimal. We pay attention to a practical special case, in which the input consists of n data records and the sets to choose from are described by conjunctions of attribute values, which we call *patterns*.

A. Motivation

To illustrate the need for the size-constrained weighted set cover problem, let us first understand the limitations of the well-studied weighed set cover problem, which has many applications ranging from facility location to workforce creation to transaction summarization [7], [8], [18]. Consider a data set of real-world entities (e.g., individuals, business listings, sales transactions), each of which is identified by an ID, described by the categorical attributes Type and Location, and associated with a numeric attribute Cost. Table I shows an example with $n = 16$ entities. Here, the sets of entities correspond to data cube patterns on the attributes Type and Location, and specify workforce groups of individuals, marketing campaigns to businesses, summaries of sales transactions, etc. For example, the pattern {*Type=ALL, Location=West*} covers records 1 and 7 in Table I, while the pattern {*Type=B, Location=South*} covers records 3 and 13. Further, the cost (weight) of the set/pattern is computed as a function of the costs of the entities in the set in an application-specific way, and corresponds to the cost of deploying a workforce group, doing a marketing campaign, etc. For illustrative purposes, we use the maximum cost of the entities in a set as the cost of the set. Table II shows all the possible patterns for the entities data from Table I, along with their costs and the number of entities covered by the pattern (referred to as Benefit). We will refer to these patterns by their pattern numbers in the rest of this paper.

TABLE I: Real-world entities data set

ID	Type	Location	Cost
1	A	West	10
2	A	Northeast	32
3	B	South	2
4	A	North	4
5	B	East	7
6	A	Northwest	20
7	B	West	4
8	B	Southwest	24
9	A	Southwest	4
10	B	Northwest	4
11	A	North	3
12	B	Northeast	3
13	B	South	1
14	B	North	20
15	A	East	3
16	A	South	96

The (partial) weighted set cover problem seeks to cover a specified fraction of the entities using a collection of sets with the minimum sum of costs (weights). In the applications mentioned, it may result in the cheapest set of marketing campaigns that reach a specified fraction of businesses, or identify the cheapest set of summaries that contain a specified fraction of sales transactions. Let the desired fraction of entities to be covered in our example be $\frac{9}{16}$.

The solution to the partial weighted set cover problem would return the 7 sets/patterns P3, P5, P6, P8, P10, P12, P13, with a total cost of 24. While this is the cheapest solution to the problem, the number of sets returned is a large fraction of the size of the data set, and may be too many for the application, which may want to specify an upper bound on the number of sets returned, e.g., the number of facility locations, or marketing campaigns to businesses.

This additional constraint on the number of returned sets is precisely what is specified by our problem. In particular, the

TABLE II: All possible patterns in the entities data set

Pattern no.	Type	Location	Cost	Benefit
P1	A	West	10	1
P2	A	Northeast	32	1
P3	A	North	4	2
P4	A	Northwest	20	1
P5	A	Southwest	4	1
P6	A	East	3	1
P7	A	South	96	1
P8	B	South	2	2
P9	B	East	7	1
P10	B	West	4	1
P11	B	Southwest	24	1
P12	B	Northwest	4	1
P13	B	Northeast	3	1
P14	B	North	20	1
P15	A	ALL	96	8
P16	B	ALL	24	8
P17	ALL	North	20	3
P18	ALL	South	96	3
P19	ALL	East	7	2
P20	ALL	West	10	2
P21	ALL	Northeast	32	2
P22	ALL	Southwest	24	2
P23	ALL	Northwest	20	2
P24	ALL	ALL	96	16

size-constrained weighted set cover problem seeks to cover (at least) a specified fraction of the entities using a collection of (at most) a specified number of sets with the minimum sum of costs (weights). If $k = 2$ is the constraint on the number of sets to be returned while covering a fraction of $\frac{9}{16}$ entities, the optimal solution consists of sets P6 and P16, with a total cost of 27. While this solution is a little more expensive than the solution returned by weighted set cover, it is a much smaller solution, which may be desirable in a number of applications.

Note that if we wanted the cheapest solution with $k = 2$ sets, without a constraint on the number of entities covered, the solution would consist of P6 and P8, which cover only a fraction of $\frac{3}{16}$ entities. Also, if we wanted any solution with $k = 2$ sets, and a $\frac{9}{16}$ coverage requirement, the solution returned (e.g., P11 and P15) has a high cost (of 120).

Thus, the novelty of our problem is that we need to simultaneously optimize for high coverage, low sum of costs and small size (at most k sets). While satisfying two out of these three goals is known, it is not sufficient for our problem.

To the best of our knowledge, this problem has not been studied before despite having many applications. In facility planning, a city may need to find the best locations for hospitals, in order to minimize the total construction cost and ensure that a desired fraction of the population is close to at least one location. Due to staff size limits or zoning constraints, at most k such objects may be built. Similarly, there are applications in video/ blog recommendation systems [18], variants of facility location [7] and workforce creation problems [8], where set cover appears naturally, and having both size and weight constraints is useful.

B. Contributions and Organization

The first contribution of this paper is a complexity study of our problem. We show inapproximability results when any one of the three required conditions must be satisfied exactly. Theorem 2 summarizes the complexity results for a general set system, while Theorems 1 and 3 contain the corresponding results for the special case when the sets are patterns.

As our second contribution, we propose two approximation algorithms that may select slightly more than k sets, and cover slightly fewer elements than desired, and have a sum of weights that is slightly higher than optimal. The first algorithm, *cheap max coverage* (CMC), finds a solution whose cost (sum of weights) is within a logarithmic factor of optimal, and uses no more than $(1 + \epsilon)k$ sets to cover a $(1 - \frac{1}{e})\hat{s}$ fraction of the elements. This algorithm is a non-trivial adaptation of the well-known greedy heuristic for *maximum coverage*. Theorem 5 summarizes the guaranteed performance bounds obtained by CMC. In Section III we illustrate why natural extensions of *maximum coverage* or its budgeted version do not work for our purpose. The second algorithm, *concise weighted set cover* (CWSC), is an adaptation of the *weighted set cover* heuristic. While it chooses at most k sets, it does not come with approximation guarantees. In practice, it performs very well both in terms of approximation and running time.

Our third contribution is a set of performance optimizations of CMC and CWSC for the special case of patterned sets. We exploit the lattice structure in this case to prioritize patterns based on their absolute coverage provided that their costs are low (see Figures 5 and 6 in Section VI).

C. Roadmap

The remainder of this paper is organized as follows. Section II formally defines our problem. Section III discusses prior work and explains why existing approaches cannot solve our problem. Section IV proves the hardness of approximating our problem. Section V presents our solutions, with performance optimizations for patterned sets given in Section V-C. Section VI presents experimental results, including significant performance benefits of our optimizations, a comparison of CWSC and CMC, and a comparison to the standard maximum coverage and weighted set cover heuristics. Section VII concludes the paper.

II. Problem Statement

Table III lists the symbols used in this paper. The input to our problem is a collection of elements T and a collection of sets defined over T, including a set that covers all the elements. For the special case of patterned sets, T is a data set with j pattern attributes, D_1 through D_j, and possibly other categorical or numeric measure attributes. Each pattern $p \in (dom(D_1) \cup \{\text{ALL}\}) \times \cdots \times (dom(D_j) \cup \{\text{ALL}\})$ is defined as having, in each attribute D_i, a value from $dom(D_i)$ or else a wildcard symbol 'ALL'. A record $t \in T$ matches pattern p if t and p agree on all the non-wildcard attributes of p, that is, $p[D_i] = \text{ALL}$ or $t[D_i] = p[D_i]$, for $1 \leq i \leq j$. Attribute tree hierarchies or numerical ranges may be used as well, but are not considered in this paper.

The output is a collection of up to k sets, denoted S. Each set $s \in S$ covers one or more elements; let $Ben(s)$ be these

TABLE III: List of symbols used in this paper

T	A collection of elements given as input
D_i	The i-th pattern attribute
$Dom(D_i)$	Active domain of the i-th pattern attribute
j	Number of pattern attributes
S	A collection of sets returned as output
s	A set
p	A pattern
k	Maximum number of sets in a solution
$\hat{s}$	Fraction representing desired coverage

elements. In the special case, each set corresponds to a pattern p, and the *benefit set* of p, denoted $Ben(p)$, is the set of records from T covered by p. The *benefit*, or coverage, of p is the number of records from T matching p, or $|Ben(p)|$. We will use the terms benefit and coverage interchangeably.

Each set s is associated with a cost or weight, denoted $Cost(s)$, that is given as part of the input. The details of computing set (or pattern) weights are orthogonal to our algorithms. In the example in Section 1, we used a simple cost function $Cost(p) = \max_{t \in Ben(p)} t[Cost]$.

Suppose that we have already selected one or more sets into S. The *marginal benefit set* of a new set s given S, $MBen(s, S)$, is the set of elements from T covered by s that have not yet been covered by an existing set in S. Furthermore, we define $Gain(s) = \frac{|Ben(s)|}{Cost(s)}$ and marginal gain: $MGain(s, S) = \frac{|MBen(s,S)|}{Cost(s)}$.

We now state the problem that we will solve in this paper:

Definition 1: Given a collection of elements T, a collection of sets over T (including a set that covers all the elements in T), with each set s having a non-negative weight $Cost(s)$, an integer k, and a coverage fraction $\hat{s}$, the size-constrained weighted set cover problem is to find a sub-collection of sets S of size at most k such that $|\cup_{s \in S} Ben(s)| \geq \hat{s}|T|$ and $\sum_{s \in S} Cost(s)$ is minimal.

For the case of patterns, the pattern with ALL in every attribute is the set that covers all the elements. This ensures that there always exists a feasible solution to our problem regardless of the input.

III. Related Work

As mentioned earlier, in our problem we need to simultaneously optimize for coverage, cost (sum of weights) and solution size (at most k sets). Previous work on set covering optimizes either coverage and cost or coverage and size and therefore cannot solve our problem, e.g., weighted set cover (optimizes cost for fixed coverage), maximum coverage (optimizes coverage for fixed size) and budgeted maximum coverage (optimizes coverage for fixed cost) [11].

There has been work on covering a data set using few patterns [1], [4], [9], [14], [24]. However, these methods optimize for coverage and size but not cost, e.g., they may return k patterns whose union covers the most records, but do not solve our problem.

A solution that focuses on coverage and cost, but not size, was proposed in [13], in the context of item sets. Their objective was to maximize the information content of the discovered patterns subject to an upper bound on cost (description length) without constraining the number of patterns.

In budgeted maximum coverage, we need to cover the most elements within a fixed budget on the total sum of weights of the chosen sets. A greedy algorithm for this problem selects sets according to how many uncovered elements they cover divided by their weight [11]. While it may seem that this algorithm could be extended to solve our problem by stopping it after it chooses $O(k)$ sets, examples can be constructed showing that its coverage may be very low. Suppose we want to return at most ck (for constant $c \geq 1$) sets where an optimum solution can pick only k sets. We have a 100% coverage requirement. We have elements $1, 2, ..., Ck$, sets $\{1\}, \{2\}, ..., \{ck\}, \{1, 2, ..., C\}, \{1+C, 2+C, ..., 2C\}, ..., \{1+(k-1)C, ..., kC\}$. The first ck sets each cover 1 element and have weight 1, whereas the last k sets each cover C elements and have weight $(C + 1)$. Suppose $c << C$. The algorithm of [11] if allowed to pick ck sets will only pick the single cardinality sets and will have coverage ck, whereas the optimum solution will pick the last k sets to have 100% coverage. Therefore, the adaptation of [11] can have arbitrarily small coverage compared to an optimum solution.

A natural technique for solving the *weighted set cover problem* is to model it via an *integer linear program*, consider its linear relaxation and then round the fractional solution to a nearby integer optimum [22]. However, to obtain a guaranteed performance with high probability (or a deterministic algorithm) using this approach may violate the cardinality constraint by more than a $(1 + \epsilon)$ factor unless k is large.

There is a large body of work on pattern mining, but it focuses mostly on finding a representative set of patterns that may be used to estimate the frequencies (or other statistics) of other patterns; see, e.g., [17], [20], [21], [25]. However, our optimization criterion of minimizing cost while reaching a certain coverage fraction is different and therefore existing algorithms are not applicable. Another popular objective is to find a set of patterns that are independent or surprising with respect to one another; this helps avoid redundant patterns and reduces the size of the solution [16], [19], [23]. Again, our goal is different because we require patterns with high coverage and low cost, regardless of how surprising they are.

Algorithmic frameworks have been proposed to find the k best patterns according to various utility functions [3], [12]. However, it is not clear how to extend these to our problem and formulate a utility function that combines our notions of cost and coverage. Instead, we directly optimize for each of cost and coverage, while limiting the number of patterns to k.

The *AlphaSum* technique [5] optimizes for size, coverage and cost. It produces a summary with k *non-overlapping* patterns that cover the entire data set and have the lowest cost, which corresponds to the size of the domain of the attribute that has been summarized. For example, a pattern having "age=Teenager" represents tuples with ages between 13 and 19, so its cost is seven. Thus, in addition to limiting the solution to non-overlapping patterns, the definition of cost used in [5] is different.

IV. Complexity Analysis

We now prove that our problem is NP-hard and hard to approximate, even for the special case of sets defined by patterns, and computing pattern weights from the values of some numeric attribute over the records covered by the pattern. We first consider a modified version of our problem, in which we must choose the fewest patterns from those whose costs are below a given threshold so as to cover a desired fraction of the data (i.e., we do not directly optimize for cost).

Lemma 1: Given T, $\hat{s}$ and an integer τ, suppose we want to find a smallest set of patterns S over T, all of whose costs are at most τ, such that $|\cup_{p\in S} Ben(p)| \geq \hat{s}|T|$. This problem is NP-hard and hard to approximate to within any constant less than 34/33.

Proof: We show a polynomial-time reduction from VERTEX COVER IN TRIPARTITE GRAPHS, which is not only NP-Complete but which cannot be approximated to within any constant smaller than $\frac{34}{33}$ assuming $P \neq NP$ [2].

Let G be a tripartite graph with vertex partition (A, B, C) having, say, m edges. Denote the vertices of A, B and C as a_i, b_j and c_k, respectively. Create three new vertices x, y, z which are not vertices of G. Build a data set T with records with pattern attributes D_1, D_2, D_3, where $dom(D_1) = A \cup x, dom(D_2) = B \cup y, dom(D_3) = C \cup z$, as well as a measure attribute M that will be used to compute pattern weights. Given edges e of G, populate the data set such that (1) if $e = \{a_i, b_j\}$, add record $(a_i, b_j, z|\tau)$; (2) if $e = \{a_i, c_k\}$, add record $(a_i, y, c_k|\tau)$; and (3) if $e = \{b_j, c_k\}$, add record $(x, b_j, c_k|\tau)$; where $'|'$ emphasizes that the number following $|$ is the corresponding value of the measure attribute M. Finally, add one more record $(x, y, z|W)$ for some $W > \tau$. Define $\hat{s} = \frac{m}{m+1}$; that is, the set of patterns S should cover at least m out of $m + 1$ records in T. Assign the cost of a pattern as the maximum value of the measure attribute of all the records that it covers. This completes the reduction; clearly, it can be done in polynomial time.

Consider any set of patterns S that satisfies the coverage fraction $\hat{s}$. We prove that a feasible solution must contain only patterns of the form $(a_i, ALL, ALL), (ALL, b_j, ALL)$ and (ALL, ALL, c_k). Note that the following patterns each have cost $W > \tau$ and hence cannot be part of S: (ALL, ALL, ALL), $(x, \text{ALL}, \text{ALL})$, $(\text{ALL}, y, \text{ALL})$, $(\text{ALL}, \text{ALL}, z)$, (x, y, ALL), (ALL, y, z), (x, ALL, z), (x, y, z). Therefore, S cannot cover $(x, y, z|W)$. Since S must cover m out of $m + 1$ records of T, all records excluding (x, y, z) must be covered by S. Note that there is a feasible solution: just take all the patterns $(a_i, ALL, ALL), (ALL, b_j, ALL)$ and (ALL, ALL, c_k). We will show that the smallest set of patterns in S that cover the desired fraction of the data equals the size of the smallest vertex cover.

Consider any pattern p in the set S. It must match at least one record in T. Say p matches $(a_i, b_j, z|\tau)$, the other two cases of matching $(a_i, y, c_k|\tau)$ or $(x, b_j, c_k|\tau)$ being symmetric. This means that the first component of p is either a_i or ALL, its second component is either b_j or ALL, and its third component is either z or ALL. However, both of its first two components cannot be ALL, since then p will have cost W and cannot be in S. So let us assume by symmetry that the first component is a_i. Now replace that pattern p by $p' = (a_i, ALL, ALL)$. The new pattern covers at least as many records of T as the previous one, and has a cost of τ.

By repeating this process, we can create a pattern set S, all of whose patterns have cost τ, and whose coverage is at least as large; hence, the coverage is exactly m. Furthermore, since x, y, z appear nowhere in S, for each edge $\{a_i, b_j\}$, we must have chosen either (a_i, ALL, ALL) or (ALL, b_j, ALL); that is, we must have chosen one of the two vertices covering the edge $\{a_i, b_j\}$. Hence, we have a vertex cover whose size is at most the number of patterns in S. Therefore, the size of the smallest vertex cover is at most the number of patterns in S.

Given a vertex cover $A' \cup B' \cup C'$, $A' \subseteq A, B' \subseteq B, C' \subseteq C$, the set of patterns (a_i, ALL, ALL) for all $a_i \in A'$, (ALL, b_j, ALL) for all $b_j \in B'$, and (ALL, ALL, c_k) for all $c_k \in C'$ satisfies the coverage requirement and has the same size as the vertex cover. Hence, the size of the smallest pattern set S is at most the size of the smallest vertex cover. ■

Lemma 1 extends to cases where pattern weights are computed using other functions over the measure attribute, such as the sum or l_p-norm, as long as W is sufficiently large.

We can now prove the following result about our problem.

Theorem 1: Given k and $\hat{s}$, the size-constrained weighted set cover problem is NP-hard for the special case of sets defined by patterns. Assuming $P \neq NP$, it is not possible to obtain a polynomial-time algorithm to find a collection of patterns S, such that $|\cup_{p\in S} Ben(p)| \geq \hat{s}|T|$ and the number of patterns selected is at most 34/33 times k, and $\sum_{p\in S} Cost(p)$ is at most α times the optimal total cost, for any $\alpha \geq 1$.

Proof: It is easy to see that our problem is in NP: given a collection of k patterns, we can verify that they have the right total benefit and cost. To prove the NP-hardness and inapproximability claims, we give a polynomial time reduction from the problem mentioned in Lemma 1. Consider an instance of problem in Lemma 1. For each pattern that has a cost above τ, we set its new cost to ∞; else, we set the new cost to its previous value. Everything else remains the same.

If there exists a pattern set with at most 34/33 times k patterns and a total cost that is at most α times the optimal cost, then none of the patterns with cost ∞ are chosen. This implies that every selected pattern costs no more than τ. We thus have a solution to the problem from Lemma 1 using at most k times 34/33 patterns, which is a contradiction. ■

From the above theorem, the following corollary follows

Corollary 1: Assuming $P \neq NP$, it is not possible to obtain a polynomial-time algorithm to find a set S of patterns such that $|\cup_{p\in S} Ben(p)| \geq \hat{s}|T| - C$ for any constant $C \geq 1$, and the number of patterns selected is k, and $\sum_{p\in S} Cost(p)$ is at most α' times the optimal total cost, for any $\alpha' \geq 1$.

Proof: By contradiction. Suppose we can obtain a polynomial-time algorithm to find a pattern set S such that $|\cup_{p\in S} Ben(p)| \geq \hat{s}|T| - C$ for an arbitrary constant $C \geq 0$, and such that the number of patterns selected is k (for some large enough value of k, where k is a super-constant), and $\sum_{p\in S} Cost(p)$ is at most α' times the optimal total cost, for some $\alpha' \geq 1$. After running this hypothetical algorithm once,

we still need to cover C new records in to satisfy the coverage requirement of $\hat{s}|T|$ tuples exactly.

Consider an optimal solution for constructing a cheapest set of k patterns that has the desired coverage. Such a solution always exists due to the assumption that we have a set (the one will all-ALLs) that covers all the tuples. Consider the patterns that are selected in that optimal solution but not selected by our hypothetical algorithm. Each of these patterns covers at least one new record and together they must cover at least C records. We now pick at most C patterns in non-decreasing weight order such that the marginal gain of picking a set in this order is at least one. That is, each new selected pattern covers at least one new record. Therefore, with k patterns chosen in the first round and C newly selected patterns, that is with a total of $k+C$ patterns, we can cover exactly $\hat{s}|T|$ records. The total cost of the selected patterns is at most $\alpha' + C$ times the optimal cost, since each pattern selected in the second round has a lower cost than the total optimal cost. Set $\alpha = \alpha' + C$. For a large enough k, $C < \frac{k}{33}$. Therefore, by selecting at most $\frac{34}{33}$ patterns of cost at most α times the optimal cost, we can cover $\hat{s}|T|$ tuples. This gives a contradiction to Theorem 1. ■

If the set system is arbitrary and not patterned, then the following theorem summarizes the hardness of approximation results which follow directly from the hardness of set cover (unweighted and weighted) and the maximum coverage problem [15], [6], [11].

Theorem 2: Suppose there exists an optimal solution OPT to the size-constrained weighted set cover problem with total cost (sum of set weights) W containing k sets and covering $\hat{s}n$ elements.

(a) Given standard hardness assumptions that NP does not have slightly super-polynomial time algorithms ($NP \subsetneq DTIME(n^{\log\log n})$), no true approximation is possible, that is, it is not possible to obtain an algorithm that returns in polynomial time a collection of k sets that cover $\hat{s}n$ elements and whose total cost is at most αW, for any $\alpha > 0$.

(b) Assuming $NP \subsetneq DTIME(n^{\log\log n})$, it is not possible to obtain an algorithm that returns in polynomial time a collection of at most $(\log n(1 - o(1)))\, k$ sets that cover at least $\hat{s}n$ elements and whose total cost is at most αW, for any $\epsilon > 0, \alpha > 0$. Moreover, if the algorithm has no restriction on how many sets it returns, the best approximation factor α that can be obtained in polynomial time is $\alpha = \log n$.

(c) Assuming $NP \subsetneq DTIME(n^{\log\log n})$, it is not possible to obtain an algorithm that returns in polynomial time a collection of k sets that cover at least $\left(1 - \frac{1}{e} + \epsilon\right)\hat{s}n$ elements and whose total cost is at most αW, for any $\epsilon > 0, \alpha > 0$.

The following result identifies conditions under which hardness results for patterns are as strong as for arbitrary sets.

Theorem 3: For patterned set systems, if the pattern weights are arbitrary and the number of pattern attributes can be arbitrary, then the size-constrained weighted set cover problem on patterned sets is as hard as on arbitrary set systems.

Proof: Consider any arbitrary set system $\mathcal{S}$ on n elements $1, 2, ..., n$ with a weight function $w : \mathcal{S} \to \mathbb{R}^+$. We convert it to a patterned set system as follows. Suppose we have a data set with n pattern attributes, each of which may be zero or one. For each element i, we create a record such that all but the ith attribute are set to zero, and the ith attribute has a value of one. For each set $S \in \mathcal{S}$, if $S = \{i_1, i_2, .., i_l\}$, then we create a pattern which has an 'ALL' in the i_1-th, i_2-th,...,i_l-th attribute and has a value of 0 everywhere else. We set its weight to be the same as the weight of S. Clearly, the pattern corresponding to S covers exactly the elements covered by S. The remaining patterns are given a weight of infinity so they will never be chosen. This completes the construction and gives an approximation-preserving reduction from size-constrained weighted set cover for arbitrary sets to size-constrained weighted set cover for patterned sets with arbitrary weights. ■

V. Algorithms

We now propose two approximation algorithms for the size-constrained weighted set cover problem (Sections V-A and V-B), followed by their optimizations for the special case of sets defined by patterns of attribute values (Section V-C):

1) Cheap Max Coverage (CMC) is based on partial maximum coverage (Section V-A). It is guaranteed to cover$(1 - \frac{1}{e})\hat{s}|T|$ elements using at most $5k$ sets, whose sum of weights is within a factor of $\log k$ of optimal. We then give a variant of CMC that uses only $(1 + \epsilon)k$ sets, for some $\epsilon > 0$, while being within a factor of $\frac{1}{\epsilon}\log k$ of optimal.
2) Concise Weighted Set Cover (CWSC), uses k sets to cover at least $\hat{s}|T|$ elements, but does not come with cost guarantees (Section V-B).

A. Cheap Max Coverage (CMC)

1) Algorithm Description: The CMC algorithm is based on the greedy partial maximum coverage heuristic, which selects sets with the highest marginal benefit, i.e., those which cover the most uncovered elements. To incorporate cost, we guess a value for the sum of weights of an optimal solution, call it B, and try to find a solution whose sum of weights is approximately B. We start with a small value of B (equal to the sum of weights of the k cheapest sets), which is likely going to be insufficient to cover the desired number of elements, and we try larger Bs if necessary. To avoid sets with large weights, we divide sets into levels based on their weights and we limit the number of allowed sets from expensive levels.

Figure 1 shows an initial version of CMC that may choose up to $5k$ sets (we will present a modified version that uses up to $(1+\epsilon)k$ sets shortly). The input parameters are the element collection T, the set collection C, each associated with some *Cost*, the maximum solution size k, the coverage threshold $\hat{s}$, and a parameter b that determines how much the cost budget will increase if the current budget does not yield a collection of sets with enough coverage.

The loop in lines 2-29 attempts to compute a solution for a given value of B. Lines 4-5 compute the marginal benefit of each set (initially equal to the benefit). Line 6 sets the number of elements that need to be covered; in Section V-A2, we

```
Cheap Max Coverage(T, C, k, ŝ, b):
01 B = cost of the k cheapest sets;
02 repeat
03   S = ∅; // solution
04   for each s in C
05     compute MBen(s);
06   rem = (1 − 1/e)ŝ|T|;
07   for i = 1, 2, ..., ⌊log2 k⌋
08     H_i = {s ∈ C, Cost(s) ∈ (B/2^i, B/2^(i−1)]};
09   H_⌈log2 k⌉ = {s ∈ C, Cost(s) ∈ (B/k, B/2^(i−1)]};
10   H_1+⌈log2 k⌉ = {s ∈ C, Cost(s) ∈ (0, B/k]};
11   for i = 1 to 1 + ⌈log2 k⌉
12     if i = 1 + ⌈log2 k⌉
13       k_i = k;
14     else
15       k_i = 2^i;
16     for j = 1 to k_i // pick ki sets from Hi
17       q = arg max_{s∈H_i} |MBen(s)|;
18       if q = NULL break;
19       S = S ∪ {q}; // q added to the output
20       C = C \ {q};
21       rem = rem − |MBen(q)|;
22       if rem ≤ 0
23         return S;
24       for each remaining s ∈ C
25         MBen(s) = MBen(s) \ MBen(s);
26         if |MBen(s)| = 0
27           C = C \ {s};
28   B = B ∗ (1 + b);
29 until B > total cost of all possible sets
```

Fig. 1: CMC Algorithm

explain that CMC only guarantees a coverage of $(1-\frac{1}{e})\hat{s}|T|$, so we stop after finding a value of B that gives a collection of sets whose union covers at least that many elements.

Lines 7-10 divide the sets into $1+\lceil \log k \rceil$ levels based on their costs (weights). The first level, H_1, contains sets whose costs are between $\frac{B}{2}$ and B; the second level, H_2, contains sets with cost between $\frac{B}{4}$ and $\frac{B}{2}$, and so on; the last level $\lceil \log k \rceil + 1$ contains the remaining sets with costs at most $\frac{B}{k}$. When constructing the solution, we only use 2^i sets from H_i, $i = 1, 2, \cdots, \lceil \log k \rceil$, and k sets from the last level (lines 12-15). We then run the greedy maximum coverage heuristic, choosing up to two sets from the first level H_1 that cover the most uncovered elements, up to four sets from H_2 that cover the most uncovered elements, and so on (lines 11-27). After adding a set to the output, lines 24-27 update the marginal benefit of the remaining sets and drop those whose marginal benefit is now zero. We return the current collection of sets as soon as the desired coverage is reached (lines 22-23); if we cannot reach the coverage threshold with the current cost budget B, we increase the budget by a factor of $1+b$ (line 28), and we find a new collection of sets using the new value of B (back to line 2).

We now give an example of CMC using the data set from Table I (note that CMC works for arbitrary sets, but the example uses patterns). We refer to patterns by their numbers (recall Table II). Let $k = 2$, $(1-\frac{1}{e})\hat{s} = \frac{9}{16}$, and $b = 1$.

Since the two cheapest patterns have a total cost of five, we use $B = 5$ in the first iteration. This gives H_1 with costs between 3 and 5, and H_2 with costs below three; we can choose up to two patterns from each of these two levels. Starting with H_1, we first pick pattern P3, which is the only pattern with cost between 3 and 5 that covers two records; all the others cover only one record. We then update the marginal benefits of the patterns that intersect with pattern P3 (namely P15, P17 and P24). Next, we pick either pattern P5, P6, P10, P12 or P13, depending on how we break ties (and update the marginal benefits of the remaining candidates). We have now covered 3 records. Moving to H_2, only pattern P8 is at this level, with a benefit of two, so we choose it and break out of the inner for-loop (line 18). With $B = 5$, we could only cover five records, so we need to try a larger B.

Since $b = 1$, we use $B = 10$ in the second iteration. We now have H_1 with costs between 6 and 10, and H_2 with costs below six. As before, we can choose up to two patterns from each level. Starting with H_1, we select patterns P19 and P20 (and update the marginal benefits of the remaining patterns after each selection) since they both cover two (uncovered) records each and other patterns in H_1 only cover one each. For H_2, we choose P3 and P8; they cover two records each while other patterns in H_1 cover one each. This gives a total of 8 records covered, which is not enough.

In the third iteration, $B = 20$, giving H_1 with costs between 11 and 20 and H_2 with costs below 11. From H_1 we first choose pattern P17, which covers 3 records. Next, we choose pattern P23, which covers 2 (uncovered) records. Moving to H_2, there are three patterns that cover 2 uncovered records (others cover only one): P8, P19 and P20. Choosing any two of these gives total coverage of nine and we are done.

2) Analysis: We prove the following result about CMC.

Theorem 4: If there exists an optimal collection of k sets with total cost C that covers $\hat{s}|T|$ elements, CMC returns a solution with up to $5k$ sets and total cost at most $(1+b) * (2\lceil \log k \rceil + 1) * C$ that covers at least $(1-\frac{1}{e})\hat{s}|T|$ elements.

Proof: The maximum number of sets chosen by the algorithm is $k + \sum_{i=1}^{\lceil \log_2 k \rceil} 2^i$, since we pick at most k sets from the last level and at most 2^i sets from the i-th level for $i = \{1, 2, \cdots \lceil \log_2 k \rceil$. This is at most $k + 2 * (2^{\lceil \log k \rceil} - 1)$, which is at most $k + 2 * (2^{\log k + 1} - 1)$, which is at most $5k - 2$.

To compute the total cost of the solution, note that there are exactly $\lceil \log_2 k \rceil + 1$ levels H_i and the total cost of sets selected from any level i, $i = \{1, 2, \ldots, \lceil \log_2 k \rceil$, is at most $2B$ for any given value of B. This comes from the fact that at H_i, the most expensive set has a cost of no more than $\frac{B}{2^{i-1}}$ and, we pick at most 2^i sets from it. From the last level, we pick at most k sets, each of which can cost at most $\frac{B}{k}$. Hence, the total cost of the sets chosen from the last level is at most B. Thus, the total cost is at most $(2\lceil \log_2 k \rceil + 1)B$ for a given value of B. Furthermore, since the algorithm tries different values of B that increase by a factor of $1+b$, it can "guess" the optimal cost within this factor.

The coverage guarantee follows from the fact that if there exists an optimal algorithm that covers, say, R elements using, say, l sets, then the greedy maximum coverage heuristic is guaranteed to cover $(1-\frac{1}{e})R$ elements using exactly l sets. In our algorithm, we are invoking this heuristic for every level. Specifically, suppose that we have guessed an optimal value of

B and consider the iteration of the CMC algorithm that uses this value of B or within a factor of $(1+b)$ of B. Clearly, an optimal solution can choose at most 2^i sets from level H_i without exceeding the total cost B, and it can choose at most k sets from the last level. That is exactly the maximum number of sets that CMC is allowed to choose from each level. Suppose that an optimal solution chooses some number of sets from each H_i that cover R_i uncovered elements. It follows from the approximation guarantee of the greedy maximum coverage heuristic that CMC will choose sets from each H_i that cover at least $(1-\frac{1}{e})R_i$ uncovered elements, which means that $(1-\frac{1}{e})\hat{s}|T|$ elements will be covered overall. ■

CMC can be generalized to choose $1+l$ sets from the first level, $(1+l)^2$ sets from the second level, etc., where the first level contains sets with costs between $\frac{B}{1+l}$ and B, the second-level sets have costs between $\frac{B}{1+l}$ and $\frac{B}{(1+l)^2}$, etc. The generalized CMC algorithm chooses at most $k(1+\frac{(1+l)^2}{l})$ sets whose total cost is at most $O((1+b)(1+l)\log_{1+l} k*C)$, where C is the cost of an optimal solution. Setting $l=1$, as we did in Figure 1, minimizes the number of sets selected at the expense of the total cost.

3) Reducing Solution Size: We can modify CMC to produce a nearly-optimal collection of at most $(1+\epsilon)k$ sets, for $\epsilon > 0$. Recall that CMC divides sets into levels and chooses at most two from the first level, at most four from the second level, and so on up to at most k from the last level. The trick is to merge some of the levels.

Suppose $k=12$ and $\epsilon = 0.5$, meaning that we allow at most 18 sets. As before, we reserve up to $k=12$ sets for the last level, which leaves $\epsilon k = 6$ sets for the other levels. This allows us to define H_1 (for sets with costs between $B/2$ and B) and choose two sets from it, and H_2 (for sets with costs between $B/4$ and $B/2$) and choose four sets from it; additionally, the last level, H_3, will be defined to contain sets with cost below $B/4$ and we can choose 12 sets from it. (In comparison, for $k=12$, the original CMC algorithm creates five levels and chooses up to 2, 4, 8, 16 and 12 sets, respectively, from them.)

Using the fact that $2+4+8+\ldots+2^i = 2^{i+1}-2$, we modify lines 7-16 of the original CMC algorithm as follows:

```
07   num_levels = 0;
08   for i = 1; εk ≥ 2^{i+1} − 2; i++
09      H_i = {s ∈ C, Cost(s) ∈ (B/2^i, B/2^{i−1}]};
10      k_i = 2^i;
11      num_levels++;
12   H_i = {s ∈ C, Cost(s) ∈ (0, B/2^{i−1}]};
13   k_i = k;
14   num_levels++;
15   for i = 1 to num_levels
```

The effect of the above modification on the guarantees provided by CMC is as follows.

Theorem 5: If there exists an optimal collection of k sets with total cost C that covers $\hat{s}|T|$ elements, the above modification of CMC gives a solution with up to $(1+\epsilon)k$ sets, for $\epsilon > 0$ and total cost at most $O((\frac{1+b}{\epsilon})\log k * C)$ that covers at least $(1-\frac{1}{e})\hat{s}|T|$ elements.

Proof: The algorithm creates $j+1$ levels such that $2^{j+1}-2 \leq \epsilon k < 2^{j+2}-2$. For levels $i=1,2,\ldots,j$, the algorithm picks at most 2^i sets from each level and for the $(j+1)$-th level, it picks at most k sets. Hence, the total number of sets selected is at most $k+\sum_{i=1}^{j} 2^i = k+2^{j+1}-2$, which is at most $(1+\epsilon)k$.

The sum of weights, similar to Theorem 3, is at most $2Bj + k\frac{B}{2^{j+1}}$ for a given value of B. This is at most $O(B\log \epsilon k + \frac{B}{\epsilon})$ which is $O(\frac{1}{\epsilon}B\log k)$. Since the algorithm tries different values of B that increase by a factor of $1+b$, it can "guess" the optimal cost within this factor. Hence, an additional factor of $(1+b)$ appears in the approximation guarantee for cost.

The coverage guarantee comes from the same argument as in Theorem 3. An optimal algorithm can pick at most 2^i sets from the i-th level, $i=1,2,\cdots,j$, and at most k sets from the $(j+1)$-th level without violating B. That is exactly the maximum number of sets that CMC chooses from each level. Now, we invoke the approximation guarantee of the greedy max coverage heuristic and obtain that at least $(1-\frac{1}{e})\hat{s}|T|$ elements are covered overall. ■

4) Parameter Settings: CMC requires a value for b that controls the trade-off between running time and accuracy (a higher b means that fewer candidate solutions will be created for various cost budgets B, but the total cost of the solution may be higher), and a value for ϵ which controls the trade-off between the number of sets selected and accuracy (the fewer the sets, the further away the total cost may be from an optimal solution). In Section VI, we will experimentally evaluate the impact of these two parameters.

B. Concise Weighted Set Cover (CWSC)

CMC provides approximation guarantees, but it may select more than k sets. The next algorithm—CWSC—finds at most k sets, but it does not come with approximation guarantees. We start with the partial weighted set cover heuristic of choosing sets with the highest marginal gain, which optimizes for cost and coverage. Additionally, in the first iteration we only consider sets that cover at least $\frac{1}{k}$ of the number of elements that need to be covered; in the second iteration, we only consider sets that cover at least $\frac{1}{k-1}$ of the remaining number of elements that need to be covered; and so on.

Figure 2 gives the pseudocode. Lines 3-4 compute the marginal benefit (initially equal to the benefit) of each set. Line 6 chooses the set with the highest marginal gain from those which cover enough uncovered elements; if we cannot find such a set, we return no solution (or we can return the "default" solution with the set that contains all the elements in T). In line 9, we update the number of elements that remain to be covered; in line 10, we return the current collection of sets S as soon as we have covered the required fraction of elements; and in lines 11 through 15, we update the marginal benefits of the remaining sets. Note that we can now remove sets with zero marginal benefit (lines 14-15).

For an example of CWSC, recall the data from Table I and the patterns in Table II; note that CWSC works for arbitrary sets. Again, assume $k=2$ and $\hat{s}=\frac{9}{16}$. In the first iteration, we have $k=2$ patterns that we can choose and we need to cover 9 records, so we only consider patterns that cover at least 4.5

```
Concise Weighted Set Cover(T, C, k, ŝ):
01 S = ∅; // solution
02 rem = ŝ|T|; // how many elements to cover
03 for each set s in C
04   compute MBen(s)
05 for i = k downto 1
06   q = arg max_{s∈C,|MBen(s)|≥rem/i} MGain(s);
07   if q = NULL return "No solution";
08   S = S ∪ {q}; // q added to the output
09   rem = rem − |MBen(q)|;
10   if rem ≤ 0 return S;
11   C = C \ {q};
12   for each s ∈ C
13     MBen(s) = MBen(s) \ MBen(q);
14     if |MBen(s)| = 0
15       C = C \ {s};
```

Fig. 2: CWSC Algorithm

records. There are only three such patterns: P15, P16 and P24, of which P16 has the highest marginal gain of $\frac{8}{24}$. We then update the marginal benefits of all patterns that intersect with P16 and remove those which now have a marginal benefit of zero (namely patterns P8 through P14 and P16). Since the marginal benefit of P16 was 8, in the second iteration we have one pattern remaining and only one additional record to cover. We consider all patterns with marginal benefit of at least one and choose the one with the highest gain, which is P3 (marginal gain of $\frac{2}{4}$).

Running Time Analysis for CMC and CWSC.: Here is a sketch of the running time analysis. The worst case running time of CMC is $O([\min(\log_{1+b}(W/W_{min}), m)] * [min(\hat{n}, k) \log m])$ where $\hat{n} = \hat{s}n$ is the required coverage, m is the total number of sets, W is the total weight of the sets and W_{min} is the minimum weight of any set. The first factor comes from the maximum number of guesses required; note that value of b plays a role here. If the ratio W/W_{min} is polynomially bounded, then the first factor only contributes a logarithmic increase in running time. If this ratio can be exponential, then of course the total number of guesses can be at most the total number of sets in the system. The remaining factor comes from the fact, that for each guessed optimum value, the rest can be done in a time proportional to maximum-k coverage. Analogously, the worst case running time of CWSC is $O(\min(\hat{n}, k) \log m + m \log m)$. The first factor again comes from an analysis similar to max-k coverage, and the second factor comes from the fact that we may have to discard all the sets if none of them seems appropriate for coverage requirements.

C. Optimizations for Patterned Sets

The number of possible patterns may be very large, therefore the loops in lines 4-5 of CMC and lines 3-4 of CWSC (computing the marginal benefit of every possible pattern), and the loops in lines 24-27 of CMC and lines 12-15 of CWSC (maintaining the marginal benefit of every remaining pattern after inserting a new pattern into the solution set) may be expensive. However, CWSC (in particular, the maximum coverage subroutine) and CMC both prefer patterns with high marginal benefit. Thus, for the special case of patterned sets, we can ignore patterns with low marginal benefit until they have a chance to be selected into the solution. A similar idea has appeared in [9] in the context of optimizing for size and benefit (but not cost); however, since our problem also involves cost, the technical details of our optimizations (as well as the actual algorithms) are quite different.

We define the *children* and *parents* of a pattern p as follows. To find the children of p, we replace any single wildcard *ALL* with some value from that attribute's active domain; to find the parents of p, we replace any single constant with the wildcard *ALL*. In Table I, $\{Type = A, Location = ALL\}$ has only one parent: $\{Type = ALL, Location = ALL\}$, and seven children: $\{Type = A, Location = North\}$, $\{Type = A, Location = South\}$, etc. For any pattern p, all of p's parents have a (marginal) benefit no lower than that of p; furthermore, all of p's children have a (marginal) benefit no higher than that of p. The insight behind the proposed optimizations is to make use of this property to reduce the number of patterns that need to be considered.

1) Optimized CWSC for Patterned Sets: We begin with the optimized version of CWSC, shown in Figure 3. The first optimization appears in lines 1-6: instead of enumerating all the patterns right away and computing their benefits, the candidate set C only contains the all-wildcards pattern and will be filled as the algorithm progresses. Ignoring lines 7-20 for the moment, lines 21-30 are almost the same as before, except that line 21 simply finds the pattern in C with the highest marginal gain without having to check if this pattern has enough benefit. This is related to the second optimization: at any point in time, C only contains patterns that have the required marginal benefit (of $\frac{rem}{i}$) for the current iteration; patterns whose marginal benefit is lower than that cannot be included in the output S at this time and may be safely ignored to improve efficiency. Lines 7-20 ensure that this condition is always true, as detailed below.

Lines 8-10 remove candidate patterns from the previous iteration whose marginal benefit has become too low (i.e., they must have intersected with the pattern that was added to S in the previous iteration). Line 11 initializes a "waitlist" W that contains new candidates whose children need to be examined (because those children, and perhaps even their children, might be candidates in the current iteration). Initially, the waitlist is all of C because all the current candidates might have children that are also candidates. In other words, since all the current patterns in C have marginal benefit above $\frac{rem}{i}$, it is possible that some of their children will also meet this requirement. To find new candidate patterns for the current iteration, we examine the children of existing candidates (which are not already in C or in the output S) in line 15.

Note that a child pattern is considered only if all of its parents are currently in C (a child's benefit is no higher than the benefit of any of its parents, so if a parent is missing from C, then its benefit is not high enough, and therefore the child's benefit is also not high enough). Furthermore, the "$\arg\max p \in C \wedge p \in W$" condition in line 13 ensures that we examine the waiting list in order of marginal benefit, which ensures that we examine parents before their children. The loop starting in line 12 runs until we have examined all the patterns in the waitlist W.

```
Optimized Concise Weighted Set Cover(T, k, ŝ)
01 C = ∅; // patterns to consider
02 S = ∅; // solution
03 rem = ŝ|T|; // how many tuples to cover
04 for only the all-wildcards patern p
05   compute MBen(p) and Cost(p)
06   C = C ∪ {p};
07 for i = k downto 1
08   for each p ∈ C
09     if |MBen(p)| < rem/i
10       C = C \ {q};
11   W = C;
12   while W ≠ ∅
13     q = arg max_{p∈C∧p∈W} |MBen(p)|;
14     W = W \ {q};
15       for each child pattern m of p
         that is not in C or S
16         if all parents of m are in C
17           compute MBen(m) and Cost(m);
18           if |MBen(m)| ≥ rem/i
19             C = C ∪ {m};
20             W = W ∪ {m};
21   q = arg max_{p∈C} MGain(p);
22   if q = NULL return "No solution"
23   S = S ∪ {q}; // q added to the output
24   rem = rem − |MBen(q)|;
25   if rem ≤ 0 return S;
26   C = C \ {q};
27   for each p ∈ C
28     MBen(p) = MBen(p) \ MBen(q);
29     if |MBen(p)| = 0
30       C = C \ {p};
```

Fig. 3: Optimized CWSC Algorithm for Patterned Sets

We now illustrate the optimized CWSC algorithm using the same data set and parameters as before ($k = 2$ and $\hat{s} = \frac{9}{16}$). Initially, line 6 adds pattern P24 (the all-wildcards pattern) to C, and, in the first iteration of the main loop, W only contains pattern P24 (line 11). In lines 15-16, we examine the children of P24 and add patterns P15 and P16 to C and W. Other children of pattern P24 fail the test in line 18 (their marginal benefit is below $\frac{9}{2}$), so they are not added to C or W. Next, the loop in line 12 continues with the two patterns (P15 and P16) just added to C, but none of their children pass the all-parents test in line 16. For example, we do not consider pattern P14 at this time because, while one of its parents (pattern P16) is already in C, the other (pattern P17) is not.

After three iterations of the while loop in line 12, W becomes empty and line 21 chooses P16 from the three patterns currently in C (P24, P15 and P16). Lines 27 through 30 only have to update the marginal benefit of these three patterns, whereas the unoptimized algorithm had to do this for every possible pattern.

In the second and final iteration of the main for-loop, we need to find one more pattern that covers one additional record. In line 8, C currently contains patterns P24 and P15, both of which have a marginal benefit of at least one, so they remain in C and are added to W in line 12. In line 13, we select $q =$ P24, whose marginal benefit is currently the highest in C. One of q's children is already in S, one is in C, but its other children (patterns P17 through P23) are now all added to C and W since their marginal benefits are all above one. The while loop then continues, and, in line 13, we now examine P15. Every child of this pattern has a marginal benefit of at least one and is eventually added to C.

At line 21, C contains patterns P17 through P24, P15 and P1 through P7. Pattern P3 has the highest marginal gain and is chosen as the second and final pattern in S.

Note that during every iteration of the main for-loop, when we get to line 21, C contains exactly those patterns which currently have a marginal benefit of at least $\frac{rem}{i}$. Thus, the optimized algorithm chooses exactly the same patterns (and in the same order) as the unoptimized algorithm, provided that both algorithms break ties (on marginal gain) the same way.

2) Optimized CMC for Patterned Sets: In Figure 4, we give the optimized CMC algorithm; the modified CMC algorithm that chooses at most $(1 + \epsilon)k$ patterns may be optimized in similar fashion. As in the optimized CWSC, the first modification is that the candidate set C initially contains only the all-wildcards pattern (lines 11-13). Next, rather than selecting patterns level-by-level, we consider the whole search space, starting from the all-wildcards pattern and working our way down the pattern hierarchy. We maintain two new variables: a count array (lines 15-16) that counts the number of patterns chosen from each level, and a set V of patterns that have been "visited" but not selected into S (because their cost is above B or we have already selected the maximum allowed number of patterns from their level).

The inner while-loop (lines 17-35) constructs a possible solution for the current value of B and runs until we have chosen the allowed number of patterns or examined all the candidate patterns. At every step, we take a pattern from the candidate set C with the highest marginal benefit (line 18). If its level is not NULL (i.e., its cost is below the current value of B) and we have not already selected k_i patterns from its level, we add this pattern to S (line 22) and update the marginal benefits of the remaining patterns in C (lines 24-29). We do not maintain the marginal benefit of all the possible patterns at every step, only those which are currently in C. Now, if we cannot add the pattern with the highest marginal benefit to S (because its cost is above B or because we have already chosen enough patterns from its level), we add this pattern to the visited set V (line 31) and we consider adding its children to C (lines 32-35). We insert a child pattern into C only if all of its parents have already been visited.

Since the optimizations of CMC are similar to those for CWSC (consider parents before children; instead of keeping track of the marginal benefit of every pattern, do so only for a selected set of candidates), we omit a worked example of optimized CMC.

VI. Experiments

We now evaluate the efficiency and accuracy of CMC and CWSC. The algorithms were implemented in C++. We conducted the experiments on a server with a Quad-Core AMD Opteron Processor 8356 containing 128GB of memory.

We use a real data set with pattern attributes and a numeric attribute to test the efficiency and accuracy of our algorithms.

```
Optimized Cheap Max Coverage(T, k, ŝ, b):
01 B = cost of the k cheapest patterns;
02 for i = 1 to 1 + ⌈log2 k⌉
03   if i = 1 + ⌈log2 k⌉
04     k_i = k;
05   else
06     k_i = 2^i;
07 repeat
08   S = ∅; // solution
09   C = ∅; // patterns to consider
10   V = ∅; // visited patterns
11   for only the all-wildcards patern p
12     compute Cost(p) and MBen(p);
13     C = C ∪ p;
14   rem = (1 − 1/e)ŝ|T|;
15   for i = 1, 2, ..., 1 + ⌈log2 k⌉
16     count[i] = 0;
17   while C ≠ ∅
     and Σ_{j=1}^{1+⌈log2 k⌉} count[j] ≤ Σ_{j=1}^{1+⌈log2 k⌉} k_i
18     q = arg max_{p∈C} |MBen(p)|;
19     C = C \ {q};
20     i = level of q;
21     if i ≠ NULL and ++count[i] ≤ k_i
22       S = S ∪ {q}; // q added to the output
23       rem = rem − |MBen(q)|;
24       if rem ≤ 0
25         return S;
26       for each remaining p ∈ C
27         MBen(p) = MBen(p) \ MBen(q);
28         if |MBen(p)| = 0
29           C = C \ {p};
30     else
31       V = V ∪ {q};
32       for each child pattern m of p
33         if all parents of m are in V
34           C = C ∪ {m};
35           compute Cost(m) and MBen(m);
36   B = B * (1 + b);
37 until B > cost of all-wildcards pattern
```

Fig. 4: Optimized CMC Algorithm for Patterned Sets

This data set, called LBL[1], consists of roughly 700,000 TCP connection traces. It contains five pattern attributes (`protocol`, `localhost`, `remotehost`, `endstate`, and `flags`) and a numeric attribute denoting the session length which we use for pattern weights. We also created several synthetic data sets based on LBL to test various aspects of our algorithms, as detailed below.

A. Scalability of Proposed Optimizations

First, we evaluate the scalability of the optimizations proposed in Section V-C for patterned sets, with respect to data size, the number of attributes, the maximum solution size k and the coverage threshold $\hat{s}$. We set $k = 10$ and $\hat{s} = 0.3$ unless otherwise noted. For CMC, we set b and ϵ to one for now (we will describe the effect of these parameters on the accuracy and efficiency of CMC shortly).

Figure 5 plots the running time of the unoptimized and optimized versions of CMC and CWSC as a function of data size; we varied the data size by randomly sampling from the LBL data set. Even for small data sizes, the optimized algorithms are twice as fast as their unoptimized counterparts. The running time of the optimized algorithms grows sub-linearly with the data size. As expected, CWSC is faster than CMC since CMC needs to create multiple candidate solutions for various values of the budget threshold B before it finds a feasible solution (recall Section V-A).

[1] http://ita.ee.lbl.gov/html/contrib/LBL-CONN-7.html

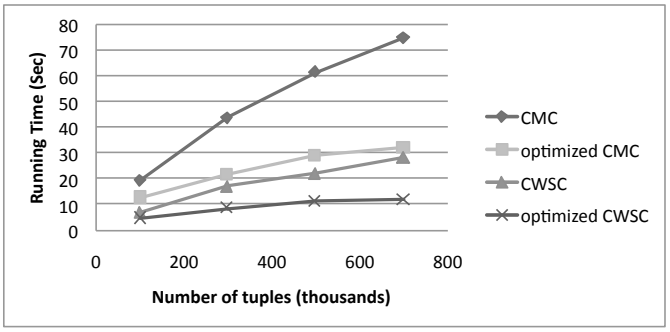

Fig. 5: Running time vs. data size

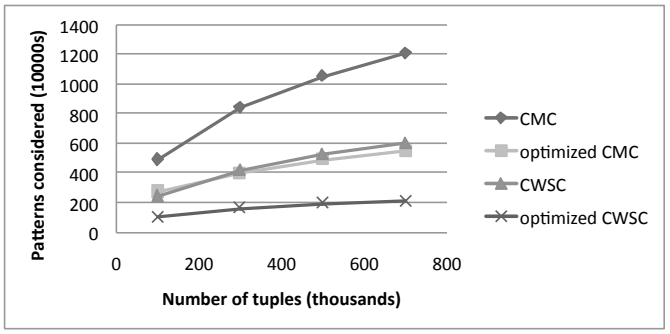

Fig. 6: Number of patterns considered vs. data size

Figure 6 explains why our optimizations improve performance: the number of patterns considered is much smaller. Again, the effect of the proposed optimizations grows as the number of tuples increases. Note that for CMC, the number of patterns considered is the sum of the patterns considered for each value of B. This is why the curve for CMC is above that for CWSC even though they use the same data set.

In Figure 7, we show the scalability of the optimized algorithms with respect to the number of pattern attributes. We performed this experiment by removing one pattern attribute at a time from the LBL data set. As before, the optimized algorithms outperform their counterparts, especially as the number of attributes increases. Again, this is because the optimized algorithms consider fewer patterns when computing the solution.

As Figure 8 shows, the maximum number of patterns allowed in the solution, k, has an interesting effect on performance. For CWSC, the running time increases as k increases since it takes more iterations to compute the solution. For CMC, the running time decreases as k increases. This is because at small values of k, it is harder to find a small number of patterns with low cost and high benefit, so CMC

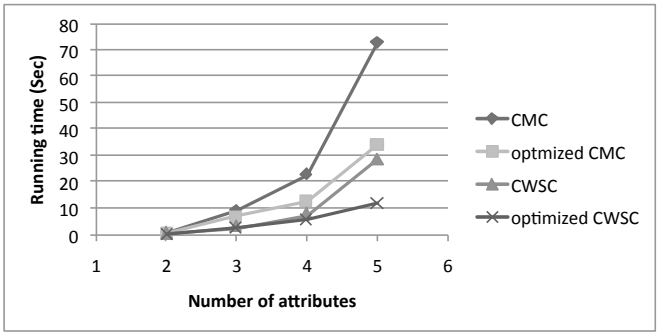

Fig. 7: Running time vs. the number of attributes

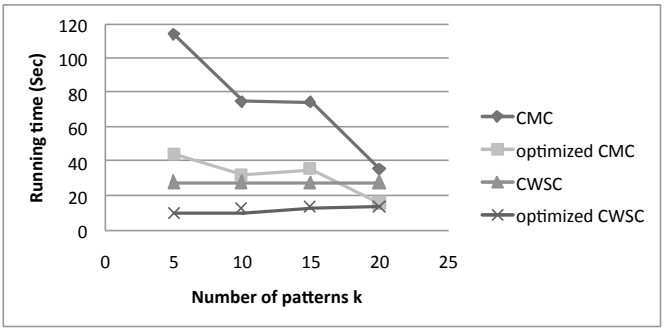

Fig. 8: Running time vs. maximum number of patterns allowed

tries many values of B before it finds a feasible solution. As k increases, it becomes easier to find a solution with lower cost, i.e., at a lower value of the budget threshold B. As before, the optimized algorithms outperform the standard ones by a factor of two or more.

Figure 9 shows that increasing the coverage fraction $\hat{s}$ does not affect the running time of CWSC, but it affects the running time of CMC; furthermore, the optimized algorithms continue to outperform the non-optimized ones by at least a factor of two. For CWSC, this makes sense since the number of iterations only depends on the maximum number of patterns, and increasing $\hat{s}$ simply means that CWSC will select patterns with higher coverage in each iteration. On the other hand, the running time of CMC increases as $\hat{s}$ increases because it becomes harder to find a feasible solution which covers that many tuples and has a cost of at most B, and therefore CMC must increase B and recompute a solution more times.

B. Comparison of CMC and CWSC

In this section, we directly compare CWSC and CMC. Previous experiments showed that CWSC is more efficient; we now show that CWSC performs as well as CMC in practice in terms of the total cost of the solution. Table IV shows the total costs of solutions obtained by CWSC and CMC (with various values of b and ϵ) for $k = 10$; we observed the same trends for other values of k. Even when the total cost of the solutions obtained by CWSC and CMC was slightly different, most of the patterns were identical. In Table V, we list the corresponding running times (in seconds), showing that CWSC takes less than half the time to run as compared to CMC with various values of b and ϵ.

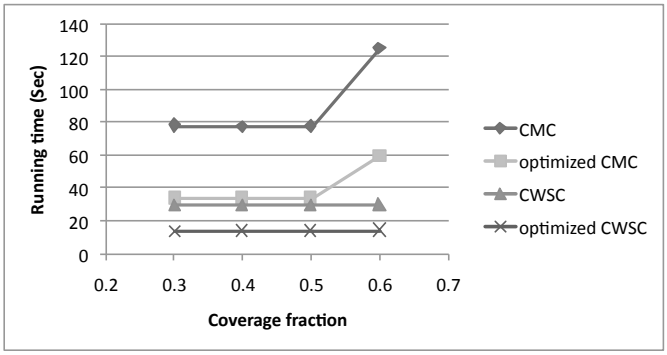

Fig. 9: Running time vs. coverage threshold $\hat{s}$

TABLE IV: Solution quality comparison of CMC and CWSC

Algorithm	$\hat{s} = 0.3$	$\hat{s} = 0.4$	$\hat{s} = 0.5$	$\hat{s} = 0.6$
CWSC	21	43	43	68
CMC ($b = \frac{1}{2}, \epsilon = 1$)	21	44	44	78
CMC ($b = \frac{1}{2}, \epsilon = 2$)	21	44	44	87
CMC ($b = 1, \epsilon = 1$)	21	44	44	75
CMC ($b = 1, \epsilon = 2$)	21	44	44	83
CMC ($b = 2, \epsilon = 1$)	24	46	46	68
CMC ($b = 2, \epsilon = 2$)	24	46	46	68

As expected, increasing b tends to increase the cost of the solution obtained by CMC, but decreases the running time.

As for the effect of ϵ, the running time of CMC increases as ϵ increases because of the overhead of maintaining more levels of patterns H_i (recall from Section 4.1 that increasing ϵ has the effect of increasing the number of levels). While varying ϵ did not have a noticeable effect on the total cost of the solution in this experiment, smaller values of ϵ gave solutions with fewer patterns (for $\epsilon = 1$, the smallest one had seven patterns, while for $\epsilon = 2$, the number of patterns was ten, which is equal to k). This is consistent with the role of ϵ, which is to trade off the number of patterns for total cost.

To further investigate the quality of solutions obtained by CWSC, we created two groups of synthetic data sets based on the LBL data set. In the first group, for each tuple, we replaced the value of the numeric attribute *session length*, call it m, by randomly selecting a value between $(1 - \delta)m$ and $(1 + \delta)m$ for different values of δ between zero and one. In the second group, we generated new values for session length from log-normal distributions with a mean of 2 (which is the logarithm of the mean of the original values) and various standard deviations between 1 and 4; we then replaced the original session lengths with the new ones in the same rank ordering. Results using these two groups of synthetic data sets were similar to those reported in Table IV: CWSC continued to return solutions whose total costs were no greater than those of CMC with various values of b and ϵ.

Also, the optimized versions of CMC and CWSC continued to outperform their non-optimized counterparts by a factor of over two on the synthetic data, similar to Figures 5 through 9.

TABLE V: Running time comparison of CMC and CWSC

Algorithm	$\hat{s} = 0.3$	$\hat{s} = 0.4$	$\hat{s} = 0.5$	$\hat{s} = 0.6$
CWSC	14	15	15	16
CMC ($b = \frac{1}{2}, \epsilon = 1$)	58	58	60	59
CMC ($b = \frac{1}{2}, \epsilon = 2$)	68	70	71	70
CMC ($b = 1, \epsilon = 1$)	33	32	34	33
CMC ($b = 1, \epsilon = 2$)	38	38	38	41
CMC ($b = 2, \epsilon = 1$)	31	30	31	30
CMC ($b = 2, \epsilon = 2$)	38	37	39	41

TABLE VI: Number of patterns required by standard weighted set cover to reach various coverage thresholds

coverage fraction $\hat{s}$	0.5	0.6	0.7	0.8	0.9
number of patterns	15	18	31	43	58

C. Comparison to Existing Approaches

Recall that the size-constrained weighted set cover problem contains three constraints: we need 1) at most k patterns 2) that cover a desired fraction of the data and 3) have the lowest sum of weights. In this experiment, we verify that existing approaches which only optimize for two out of these three goals are not suitable for our problem. Table VI shows the total cost of the solution obtained by the greedy partial weighted set cover heuristic, which chooses patterns with the highest marginal gain (i.e., the number of uncovered tuples that can be covered by the given pattern divided by the cost of the pattern). This solution optimizes for coverage and cost but not conciseness. As the coverage fraction increases, this approach cannot guarantee a small number of patterns.

We also implemented a heuristic for the partial maximum coverage problem [10]. Regardless of the coverage fraction, the heuristic returned a solution with a cost of 229, which is ten times higher than our algorithms for $\hat{s} = 0.3$ and over three times as high as for $\hat{s} = 0.6$ (recall Table IV).

D. Comparison to Optimal Solution

We now compare the cost of the solution obtained by CMC and CWSC to the cost of an optimal solution. We used small samples of the LBL data set that allowed us to obtain an optimal solution using exhaustive search. CMC found an optimal solution when we used small values of b and ϵ. CWSC almost always found an optimal solution, with one exception: for $\hat{s} = 0.5$ and $k = 5$. The cost of an optimal solution was 8, which was also found by CMC for $b = 1$ and $\epsilon = 1$. However, CWSC returned a slightly worse solution with cost 9, as did CMC for larger values of b and ϵ.

VII. Conclusions

In this paper, we proposed and solved a generalized set covering problem—size-constrained weighted set cover—in which we find k sets with the lowest sum of weights whose union covers some fraction of the elements. We proved that one of the proposed algorithms—Cheap Max Coverage—is guaranteed to produce a nearly-optimal solution. We experimentally showed that the other algorithm—Concise Weighted Set Cover—is more efficient and as effective as Cheap Max Coverage in practice, though its worst-case performance is not guaranteed. Furthermore, we showed significant performance benefits of the optimized versions of the proposed algorithms for the special case of patterned sets.

One interesting direction for future work is to study an incremental version of size-constrained weighted set cover, in which the solution must be continuously maintained as new elements arrive. Another interesting problem is how to handle multiple weights associated with each set or pattern.

References

[1] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan: Automatic Subspace Clustering of High Dimensional Data. *Data Min. Knowl. Discov.* 11(1):5-33 (2005)

[2] P. Berman, M. Karpinski: On Some Tighter Inapproximability Results (Extended Abstract). *ICALP 1999*, 200–209

[3] B. Bringmann, A. Zimmermann: One in a million: picking the right patterns. *Knowl. Inf. Syst.* 18(1): 61-81 (2009)

[4] S. Bu, L. V. S. Lakshmanan, R. T. Ng: MDL Summarization with Holes. *VLDB 2005*: 433-444

[5] K. S. Candan, H. Cao, Y. Qi, M. K. Sapino: AlphaSum: Size-Constrained Table Summarization using Value Lattices. *EDBT 2009*, 96-107

[6] U. Feige: A Threshold of ln n for Approximating Set Cover. *J. ACM* 45(4): 634-652 (1998)

[7] R. Fleischer, J. Li, S. Tian, H. Zhu: Non-metric Multicommodity and Multilevel Facility Location. *AAIM 2006*: 138-148

[8] A. Gajewar, A. Das Sarma: Multi-skill Collaborative Teams based on Densest Subgraphs. *SDM 2012*: 165-176

[9] L. Golab, H. Karloff, F. Korn, D. Srivastava, B. Yu: On generating near-optimal tableaux for conditional functional dependencies. *PVLDB* 1(1): 376-390 (2008)

[10] D. Hochbaum: Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems. In Approximation algorithms for NP-hard problems, 94-143, 1997

[11] S. Khuller, A. Moss, J. Naor: The Budgeted Maximum Coverage Problem. *Inf. Process. Lett.* 70(1): 39-45 (1999)

[12] A. J. Knobbe, E. K. Y. Ho: Pattern Teams. *PKDD 2006*: 577-584

[13] K.-N. Kontonasios, T. De Bie: An Information-Theoretic Approach to Finding Informative Noisy Tiles in Binary Databases. *SDM 2010*: 153-164

[14] L. V. S. Lakshmanan, R. Ng, C. Wang, X. Zhou, T. Johnson: The Generalized MDL Approach for Summarization. *VLDB 2002*: 766-777

[15] L. Lovasz: On the ratio of the optimal integral and fractional covers. *Disc. Math.* 13:383-390 (1975)

[16] M. Mampaey, N. Tatti, J. Vreeken: Tell Me What I Need to Know: Succinctly Summarizing Data with Itemsets. *KDD 2011*: 573-581

[17] T. Mielikainen, H. Mannila: The Pattern Ordering Problem. *PKDD 2003*: 327-338

[18] B. Saha, L. Getoor: On Maximum Coverage in the Streaming Model & Application to Multi-topic Blog-Watch. *SDM 2009*:697-708

[19] G. Sathe, S. Sarawagi: Intelligent Rollups in Multidimensional OLAP Data. *VLDB 2001*: 531-540

[20] M. van Leeuwen: Maximal exceptions with minimal descriptions. *Data Min. Knowl. Discov.* 21(2): 259-276 (2010)

[21] M. van Leeuwen, A. J. Knobbe: Non-redundant Subgroup Discovery in Large and Complex Data. *ECML/PKDD* (3) 2011: 459-474

[22] V. V. Vazirani: Approximation algorithms. *Springer*, 2001.

[23] C. Wang, S. Parthasarathy: Summarizing itemset patterns using probabilistic models. *KDD 2006*: 730-735

[24] Y. Xiang, R. Jin, D. Fuhry, F. Dragan: Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. *KDD 2008*: 758-766

[25] X. Yan, H. Cheng, J. Han, D. Xin: Summarizing itemset patterns: a profile-based approach. *KDD 2005*: 314-323