Classifier Construction Under Budget Constraints (Technical Report)

Shay Gershtein
Tel Aviv University
shayg1@mail.tau.ac.il

Tova Milo
Tel Aviv University
milo@post.tau.ac.il

Slava Novgorodov
eBay Research
snovgorodov@ebay.com

Kathy Razmadze
Tel Aviv University
kathyr@mail.tau.ac.il

ABSTRACT

Search mechanisms over large assortments of items are central to the operation of many platforms. As users commonly express filtering conditions based on item properties that are not initially stored, companies must derive the missing information by training and applying binary classifiers. Choosing which classifiers to construct is however not trivial, since classifiers differ in construction costs and range of applicability. Previous work has considered the problem of selecting a classifier set of minimum construction cost, but this has been done under the (often unrealistic assumption) that the available budget is unlimited and allows to support *all* search queries. In practice, budget constraints require prioritizing some queries over others. To capture this consideration, we study in this work a more general model that allows assigning to each search query a score that models how important it is to compute its result set and examine the optimization problem of selecting a classifier set, whose cost is within the budget, that maximizes the overall score of the queries it can answer.

We show that this generalization is likely much harder to approximate complexity-wise, even assuming limited special cases. Nevertheless, we devise a heuristic algorithm, whose effectiveness is demonstrated in our experimental study over real-world data, consisting of a public dataset and datasets provided by a large e-commerce company that include costs and scores derived by business analysts. Finally, we show that our methods are applicable also for related problems in practical settings where there is some flexibility in determining the budget.

1 INTRODUCTION

Search mechanisms over large item sets are central to the operation of many companies, such as e-commerce platforms, news sites, and stock photo archives. Since users commonly express filtering conditions based on item properties that initially are not explicitly stored in a database, companies must derive these missing properties from the item's existing metadata, such as an image or a textual description, possibly also leveraging common knowledge. This is typically achieved by training binary classifiers [58], that can test whether a given conjunction of properties expressed in a user query holds for any given item. Choosing which classifiers to construct is however not trivial, since classifiers may significantly vary in the number of search queries they are useful for and their construction cost, based, e.g., on the required amount labeled data. Moreover, queries with multiple filtering conditions can be addressed by multiple combinations of classifiers, with each classifier evaluating a different subset of these conditions.

Example 1.1. To illustrate these trade-offs, consider an online platform, where users upload items for sale. Given the query "wooden table", many matching items may not be retrieved by the search engine, since users did not explicitly specify the material, as it is evident in the image. To address this, one can train a classifier that identifies wooden tables specifically or a classifier that identifies any wooden item. The classifier testing both properties simultaneously may require fewer training examples to achieve sufficient accuracy than a classifier for all wooden items, as there is much less variability in the features of tables. On the other hand, the "wooden" classifier, while costlier, is also useful for queries involving other wooden items. Moreover, if some tables are not assigned explicitly to a "tables" category (or items such as "table covers" are erroneously assigned to this category), one may also need to complement the "wooden" classifier with a "table" classifier.

Existing solutions. Previous work on this setting [18, 22, 23] studied a model where given a query log and (estimated) construction costs of classifiers, one seeks a classifier set that can derive the results sets of all the queries, such that the overall construction cost is minimized. This model, however, is based on the often-unrealistic assumption that the budget is unlimited and allows to support *all* search queries. In practice, training each classifier is typically expensive, as it requires humans to label a large volume of high-quality training data. Thus, when the human or monetary resources are insufficient to construct classifiers that compute result sets for all queries, companies must prioritize more frequent/important queries for which there are economical classifiers.

Example 1.2. To illustrate these considerations, continuing with the above example of an e-commerce platform, consider in addition to the "wooden table" search query, also the queries "round table" and "running shoes". Companies periodically allocate a given budget for improving search engine performance, and in particular search query results, and in this toy example, it may be the case that the budget is insufficient to cover the cost of any classifier set that can compute the results sets for all three queries.

For instance, constructing a classifier that identifies running shoes may require more effort (and, thus, more money) than the classifiers for the table queries, since it is, arguably, harder to deduce from images and descriptions which shoes are suitable for running. The cost estimations might imply that the company can either construct classifiers for both table queries or only for the shoes query. In either case, at least one query would remain with an unsatisfactory result set.

Note that it is not necessarily the case that addressing the two queries is better than the single query. It may be that it is more important for the company to have improved results sets pertaining to running shoes (e.g. these may be searched for and bought more frequently), than both table queries. The prioritization of queries

is typically decided by business analysts, based on the search frequency of each query, various monetary factors, and the existing quality of result sets for different product domains. To account for this prioritization, we provide a model that allows assigning to each query a *utility* score, that reflects how important it is to compute its result set.

Model. To capture budget constraints, we study in this work an extension of the above model that includes an upper bound on the cost of the solution, which, in general, may not allow computing all queries. As queries vary in their importance, each may be assigned a *utility* score modeling the gain of constructing classifiers that compute it. We thus define the *Budgeted Classifier Construction problem* (*BCC*) of selecting a classifier set that maximizes the overall utility, without exceeding the given cost bound (we will formalize this high-level description in Section 2).

Length parameter. Before describing our results, we first define the *length parameter*, l, that crucially affects the computational complexity of *BCC*. Namely, with queries expressed as a conjunction of properties, that should hold for each item in the result set, the length parameter is defined as the maximum number of such conjuncts (properties) in any input query.

Hardness Bounds. While the non-budgeted problem of [18, 22, 23] can be solved exactly in PTIME for $l \leq 2$ and reasonably approximated for the *NP*-hard case of $l \geq 3$, we show that *BCC* is likely much harder, even when utilities and costs are uniform. Concretely, for $l = 2$, *BCC* is at least as hard as the *Densest k-Subgraph problem* (*DkS*), where one seeks a subgraph on k nodes with the maximum number of edges. The exact hardness of *DkS*, however, is unknown, and despite decades of extensive research, its best known approximation factor is $\Theta(n^{1/4})$, where n is the number of vertices, which translates to the number of distinct properties appearing in the queries of the *BCC* input. We, therefore, follow in the footsteps of works that base hardness results on this *DkS* bound [13, 16, 29] (i.e. any $o(n^{1/4})$-approximation algorithm for *BCC* would an improve on the best *DkS* algorithm). Lastly, for $l = 3$, *BCC* is as hard as the hypergraph extension of *DkS*, for which the best approximation factor is $\Theta(n^{0.62})$.

Algorithm. To offer a solution that, despite the worst-case hardness bounds above, works well in practice, we leverage the high prevalence of short queries in real-life workloads demonstrated in [23] and provide an improved algorithm for $l = 2$, which we then extend to the general case. To this end, we generalize ideas from several *DkS* works [53, 62] and combine these with novel techniques to devise a reduction from *BCC* with $l = 2$ to *DkS*. We then employ the state-of-the-art *DkS* heuristic [41], which was shown to produce solutions close to optimal, scaling even to large graphs. To further facilitate efficiency, we employ a pruning method, that can significantly reduce the size of *DkS* inputs, at the cost of a provably small additive error. We also provide a worst-case constant bound on the error incurred by our reduction. Lastly, to address the small subset of queries where $l > 3$, we devise a heuristic that allows to progressively simplify the problem, such that a larger fraction of the solution space corresponds to the case of $l = 2$, for which we have the effective algorithm above.

Experimental study. To evaluate our algorithm, we conduct an empirical study over real-world data consisting of a public dataset and private datasets provided by a large e-commerce company, that include actual costs and utility values, as estimated by business analysts. We remark that e-commerce is a particularly suitable domain for the *BCC* problem, since the most popular platforms have massive product catalogs, with insufficient information to support all search queries, as mentioned above. Consequently, e-commerce platforms devote a lot of resources to training classifiers to improve query answering [60]. Since e-commerce is a trillion-dollar industry, even modest improvements in the quality and completeness of the result sets presented to users can greatly increase profits.

The results of our evaluation demonstrate that our approach qualitatively outperforms all examined baselines for a large range of input parameters, in practical time for an offline task. We also validate the robustness of this performance over synthetic data that explores additional ranges of input parameters.

Complementary problems. In practical scenarios where there is some flexibility in the budget constraint, there are alternative objectives that may be of interest. For instance, companies may wish to maximize the ratio of utility to cost, i.e. construct a classifier set that provides maximum "bang for the buck", or, given a utility target (e.g., computing result sets for at least half of the search queries), to find the classifier set of minimum cost that reaches it (this is a direct generalization of [23], where the target was computing results to all queries). In Section 5, we show that our analysis methods can also be applied to derive complexity bounds and algorithms for these two problems.

Our main contributions can be summarized as follows.

(1) We provide a formal model for the *BCC* problem, extending previous models to capture budget constraints and variability in the importance of queries.
(2) We prove that *BCC* is *NP*-hard for any bound l on the length of the queries. For $l = 2$ and $l \geq 3$ we show that, even with uniform costs and utilities, *BCC* is at least as hard as *DkS* and its hypergraph extension, respectively, where the best known approximations are of order $poly(n)$.
(3) Despite the worst-case hardness bounds, we combine novel techniques with generalizations of existing methods to devise a practical *BCC* algorithm based on a reduction to the *DkS* problem, which we then solve via the state-of-the-art heuristic *DkS* algorithm.
(4) We present an extensive experimental study over real-world datasets, including a public dataset and private datasets obtained from a large e-commerce site, as well as over synthetic data, demonstrating the effectiveness and efficiency of our algorithm.
(5) We also show that our methods provide hardness bounds and algorithms for two related problems, where the goals are to find a classifier set that: (1) minimizes the cost while exceeding a given utility value; (2) maximizes the ratio of utility to cost.

Paper outline. Section 2 presents formal problem definitions and useful theoretical results. The hardness bounds and the algorithm for *BCC* are provided in Sections 3 and 4, respectively. Complementary problems in the *BCC* setting are studied in Section 5. Section 6 describes the setup and results of our empirical analysis. We discuss related works in Section 7, and conclude in Section 8.

2 PRELIMINARIES

We open this section by describing the formal setting for the Budgeted Classifier Construction problem (*BCC*), and how it relates to practical settings. We then provide illustrations of problem instances and highlight important model properties. Finally, we present definitions and results that will prove useful in our theoretical analysis (Sections 3, 4 and 5).

2.1 Problem Definition

Motivating setting. As explained in the introduction, the *BCC* problem arises in practice when a company's item database is missing information necessary to derive complete result sets for search queries in a given workload. Each query's filtering condition corresponds to a conjunction of one or more *properties* that must hold for each item in the result set. To complete the missing values, companies construct binary[1] classifiers, where each classifier is characterized by a set of properties, such that it can determine whether their conjunction holds for any given item. However, constructing classifiers requires human effort, which costs money, and it may be the case that it is too expensive to construct a classifier set sufficient to answer all queries. Thus, given estimates of the utility gain of answering each query, the *BCC* optimization problem seeks a classifier set that allows answering a subset of queries of the highest total utility, without exceeding a given construction budget.

Our model is agnostic to how utility is estimated and its units of measure. The only property of utility values that is in effect is the utility ratio of two queries representing the ratio of their importance (i.e. the contribution to the objective function of covering the query). In practice, the relative importance of each query can correspond to how frequently it is submitted to the company's search engine, or to a more complex metric, that also takes into account an estimation of the size of the result set or an associated monetary gain. Similarly, the cost of each classifier (and the budget) can represent the number of labeled training examples or the monetary cost of employing domain experts or crowd workers. As these are highly correlated, any cost measure would roughly derive the same problem instance.

Input. To formally model the above setting, we denote the universe of properties of size n, the input query set of size m, and the set of classifiers one can construct by P, Q, and CL, respectively. As each query or classifier is fully captured by its corresponding set of properties, we have $Q \subseteq 2^P$ and $CL \subseteq 2^P$. For any query q, let $CL_q = 2^q \setminus \emptyset$ denote its power set excluding the empty set. This models the set of all possible binary classifiers that are relevant for q, each corresponding to a different subset of its properties. Hence, the input classifier set is $CL = \cup_{q \in Q} CL_q$, the union of the power sets of all queries (except for the empty set). For example, if the query set consists of the two queries "wooden table" and "round table", then the property set is $P = \{wooden, round, table\}$, the query set is $Q = \{\{wooden, table\}, \{round, table\}\}$, and the classifier set is $CL = \{wooden, round, table, \{wooden, table\}, \{round, table\}\}$.

To simplify notation, we use x,y and z to represent properties, denoting a query $\{x, y, z\}$ as xyz, whereas a classifier $\{x, y, z\}$, that tests for the conjunction of the same properties, is denoted by XYZ. For example, if the properties "wooden" and "table" are x and z, respectively, then the classifier that tests for wooden tables is XZ.

The utility associated with each query is represented by the function $U : Q \mapsto \mathbb{R}_+$. If, e.g., a company considers that it is twice as valuable to compute the result set of the query "round table" than of "wooden table", then the utility of the former query would be twice as large. Similarly, the cost of each classifier is represented by $C : CL \mapsto [0, \infty)$, with the budget denoted by $B \in \mathbb{R}_+$. The input for the *BCC* problem is thus the tuple $\langle Q, U, C, B \rangle$.

We note that a classifier of cost 0 implies either that it is already constructed or that the corresponding properties are fully recorded in the database (e.g., if the classifier "wooden table" is already constructed, then its cost would be zero), whereas an infinite cost implies that the classifier is omitted from consideration in advance, typically since it is deemed impractical to construct. For example, classifying whether an item is "round (and) wooden" with no additional context, may be considered impractical, as in each domain the visual features of such items may be vastly different (e.g., round wooden mirrors have only wooden frames, whereas round wooden tables are primarily wood).

Covering queries. Before defining the objective, we first need to formalize which classifier combinations are sufficient to determine the result set for a given query.

For any subset $S \subseteq CL$, we define $P(S) = \cup_{X \in S} X$ as the set of all properties appearing in classifiers in S. We say that a query q is *covered* by $S \subseteq CL$ if $\exists T \subseteq S : P(T) = q$. That is, a query is covered by a set of classifiers if it contains a subset of classifiers whose conjunction tests exactly the truth value of the conjunction of exactly the properties in the query. For example, the two classifiers "wooden table" and "round table" cover together the query "round wooden table". A set of queries covered by $S \subseteq CL$ is denoted by $Q(S)$, and the utility of S is defined as the sum of utilities of $Q(S)$.

Objective. The cost of a set of classifiers S is defined as the sum of the individual costs $C(S) = \sum_{s \in S} C(s)$. The solution space of the *BCC* problem consists of classifier sets whose cost does not exceed the budget. The objective of *BCC* is to find a classifier set of maximum utility in this solution space. More formally, we aim to compute

$$argmax_{S \subseteq CL, C(S) \leq B} \sum_{q \in Q(S)} U(q).$$

Model assumptions. We assume that the classifiers are constructed in parallel and that their construction costs are independent. While some overlaps may exist in practice, e.g., shared training examples, it is arguably not trivial to quantify these a priori. Hence, as in [23], in our model the cost of each classifier is independent, and the overall cost of a classifier set is the sum of the individual costs.

We also follow [23] in assuming that partial coverage of a query is insufficient to provide any utility, as research shows that in many cases conforming only partially to search criteria can have an even worse effect on user satisfaction than not conforming at all [31]. Moreover, as in the case of construction costs, estimating in advance the relative utility of such partial covers out of the overall utility of the query is challenging.

[1] Compared to multi-valued classifiers, which are trained towards a more general objective, binary classifiers have higher classification accuracy, and are thus preferred in practice, when accuracy is essential [58].

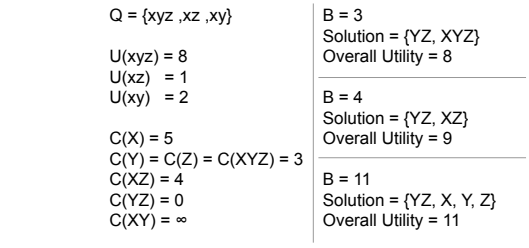


Figure 1: Three examples of *BCC* problem instances. The left side depicts the queries, utilities and costs, shared by all instances. The three different budget values, and the optimal solution corresponding to each value are depicted on the right side.

We, thus, leave to future work the study of models with more complex inputs, that account for overlaps in construction costs or partial covers that provide quantifiable value.

Length parameter. We refer to the cardinality of a query as its *length*. Let $l = l_Q$ denote the maximal length of a query in Q. This is an important parameter of the problem, as we derive different approximation bounds for the cases, $l = 1$, $l = 2$ and $l \geq 3$. In our analysis l is assumed to be a constant (in practice, it rarely exceeds 5 [27]). We use the notation $BCC_{l=i}$ to denote the *BCC* problem where $l = i$. Similarly, we define the length of a classifier as the number of properties it tests. For example, the length of the classifier XY is 2. We refer to queries and classifiers of length 1 as *singleton* queries and *singleton* classifiers, respectively.

Input size. We denote the number of queries in Q by m and the number of properties in $P = \cup_Q q$ by n. Given n, the lower bound on m is $\frac{n}{l}$ (this matches the case where all queries are disjoint and of length l), whereas the upper bound is $O(n^l)$ corresponding to the maximum number of distinct subsets of size at most $l = \Theta(1)$. Thus, m is at least of the order of n, and possibly polynomially larger.

The number of classifiers is also polynomial in n (and m). To see this, observe that CL does not include all possible classifiers corresponding to all subsets of P. For instance, if $P = \{x, y, z\}$ and $Q = \{xy, xz\}$, then $CL = \{X, Y, Z, XY, XZ\}$. The classifier YZ is not included in CL, since it is not relevant to the solution of the problem. Concretely, since no query includes both y and z, the classifier YZ cannot be used to cover any query. It follows that the number of classifiers does not exceed $m \cdot 2^l = \Theta(m)$.

The following toy example illustrates problem instances of $BCC_{l=3}$ in the above model.

Example 2.1. Consider the input in Figure 1. We will examine three problem instances over the same input, except for different budget values. The shared input, consisting of three queries, their utilities, and the costs of the seven relevant classifiers, is depicted on the left side of the figure, whereas on the right side optimal solutions are presented, corresponding to the three budget values, $B \in \{3, 4, 11\}$.

As the classifier YZ costs nothing, it can be preemptively selected into any solution. Conversely, the classifier XY of infinite cost can be omitted from consideration.

To provide a practical context, one can assume that x, y, and z are the properties "round", "wooden" and "table", respectively. Then, the classifier "wooden table" (YZ) costing nothing implies that it is already constructed. Similarly, the classifier "round wooden" costs ∞, as it is not considered practical to construct.

Instance with $B = 3$. As every query contains the property x, to cover a query one must select a classifier that also contains this property. When the budget is 3, the only valid classifier containing x is XYZ. This covers the first query xyz, as the classifier matches it exactly. With this being the only covered query, the utility of the solution $\{YZ, XYZ\}$ is 8, the utility of the covered query. Observe that the inclusion of the free classifier YZ is optional, as the solution $\{XYZ\}$ has the same utility and cost.

Instance with $B = 4$. When the budget increases to 4, of the classifiers containing x, one can also select XZ, which consumes the entire Budget. It turns out that this solution ($\{YZ, XZ\}$) improves the overall utility, as it covers both xyz and xz, whose combined utility is 9. Concretely, xz is covered by XZ, since it matches it exactly, while xyz is covered by the conjunction of $\{YZ, XZ\}$. Note that their union is exactly these three properties, and that the overlap in z makes no difference, as the conjunction xyz holds if and only if both conjunctions yz and xz hold.

Instance with $B = 11$. To improve on the previous instance, one must cover xy. It is easy to see that to cover the query xy one must select both X and Y since XY cannot be selected. Their conjunction covers xy, and when also adding the free classifier YZ, the three classifiers cover xyz. This leaves a budget of 3 to cover xz as well. The classifier XZ is too expensive, however, Z costs exactly 3, and its conjunction with X covers xz. Thus, when the budget is 11, the solution $\{YZ, X, Y, Z\}$ covers all queries, and its total utility is 11. Note that, as in the first instance, selecting YZ is optional.

Absence of costs or utilities. In some cases, it may be hard to estimate the costs in advance. In the absence of values that differentiate between classifiers, the natural compromise would be assuming uniform costs. An analogous argument applies to using uniform utilities. Moreover, to significantly reduce input size and complexity, one can restrict the solution space to singleton classifiers. This begs the question of whether the *BCC* problem becomes much easier for practical use-cases with such limitations. To this end, we show that all our hardness bounds hold, even when assuming all the aforementioned restrictions (Theorem 3.3 in Section 3), and our algorithms for the general case address these special cases with improved theoretical performance (Section 4).

2.2 Existing Results

We next present definitions and theoretical results for various problems, which we will leverage in our hardness analysis and algorithms. To simplify the presentation, we use a "soft omega" notation, $\tilde{\Theta}(\cdot)$, to hide negligible factors.

We start with the well-known *Knapsack* problem.

Definition 2.2. In the Knapsack problem, there are n items, each with a nonnegative value and weight, and a bound W. The objective is to select a subset of the items whose total weight does not exceed W, such that the sum of the item values is maximized.

Theorem 2.3. *[65] The Knapsack problem is NP-hard. However, for any $\epsilon > 0$, it admits $(1 + \epsilon)$-approximation.*

We next overview the problem studied in [23], the immediate predecessor of the present work.

Definition 2.4. In the Minimization of Classifier Construction Costs problem (*MC*3), the setting is the same as in *BCC*, except that the input does not include utilities or a budget, and the goal is to produce a classifier set of minimum cost such that it covers all of the queries.

Theorem 2.5. *[23] The MC*3 *problem where the maximum length parameter is $l = 2$ can be solved exactly in PTIME. For $l > 3$ the problem is NP-hard, and can be approximated within a $\min\{2^{l-1}, O(\log n)\}$ factor.*

A more central role in our analysis is played by graph and hypergraph density problems, as defined next (recall that hyperedges in a hypergraph correspond to node subsets of cardinalities that may exceed 2).

Definition 2.6. In the *Densest k-Subgraph problem* (*DkS*), given a graph on n nodes and an integer k, the goal is to find a subgraph on k nodes with the highest number of edges. The extension where edges have positive weights and the goal is to maximize the sum of edges weights in the subgraph is the *Heaviest k-Subgraph problem* (*HkS*). Further generalizing *HkS* to have node costs, and replacing the cardinality bound k with a total cost bound B, is the *Quadratic Knapsack problem* (*QK*). Finally, the *Densest k-Subhypergraph problem* (*DkSH*) asks for a subhypergraph of k nodes with the maximum number of hyperedges.

The following is known of the approximation hardness of *DkS* and its generalizations.

Theorem 2.7. *[6, 7, 62] All four problem in Definition 2.6 are NP-hard, even when all node degrees equal* 3*. Additionally, DkS and HkS can be approximated within a $\tilde{O}(n^{1/4})$ factor. For QK this factor increases to $\tilde{O}(n^{0.4})$. Finally, DkSH with hyperedges of size* 3*, admits $\tilde{O}(n^{0.62})$-approximation.*

Conjectured hardness of DkS. Despite decades of study [19, 39], there remains a large gap between the proven hardness of *DkS* and the best known approximation factor, $\tilde{O}(n^{1/4})$ [7]. Under widely-believed complexity assumptions, it cannot be approximated to within a constant factor [3]. There are also stronger superlogarithmic bounds derived under stronger assumptions [47]. Nevertheless, we follow in the footsteps of works that reduce the hardness of examined problems to the hardness of *DkS* [13, 16, 29], as the "Dense vs Random" conjecture [17, 48] implies that the $\tilde{O}(n^{1/4})$ factor is tight. The discussion above also roughly applies to *DkSH*, where there are somewhat stronger hardness results[4].

3 HARDNESS RESULTS

In this section, we provide approximation hardness bounds showing that *BCC* is *NP*-hard for any l and may become much harder to approximate as l increases.

We first show a simple equivalence between $BCC_{l=1}$ and the Knapsack problem (Definition 2.2), implying that Theorem 2.2 applies to $BCC_{l=1}$ as well.

Theorem 3.1. *The $BCC_{l=1}$ problem is equivalent to the Knapsack problem.*

Proof. Observe that for $l = 1$ each query is a single property, and the only classifier that can cover it also corresponds to this property. The equivalence is implied immediately by considering each item, x, in the Knapsack context, as a (singleton) query x, with its value and weight corresponding to the utility of x and the cost of X, respectively (and the weight bound W is the budget B). □

For $BCC_{l=2}$, we prove that it generalizes *DkS* (Definition 2.6), and is thus is at least as hard. In particular, following the discussion in Section 2, any $o(n^{1/4})$-approximation algorithm for *BCC* (recall that $n = |P|$ is the number of properties), would imply an improvement over the best known *DkS* algorithm. Similarly, $BCC_{l=3}$ generalizes *DkSH* with edges of cardinality 3 (Definition 2.6), for which the best known approximation factor is $\Omega(n^{0.62})$. Finally, since *BCC* can only become harder as l increases (e.g., adding one query that increases l, and of otherwise negligible effect, would retain the same hardness), all bounds also apply when $l > 3$.

We next define the special cases of *BCC* that are equivalent to *DkS* and *DkSH*.

Definition 3.2. Let I_l denote the *BCC* problem restricted to inputs where all queries are of length l, all utilities and costs of singleton classifiers are 1, all other classifier costs are ∞, and the budget is an integer.

Considering these I_l variants implies the following result.

Theorem 3.3. *The BCC problem with $l = 2$ and $l \geq 3$ is at least as hard as DkS and DkSH, respectively. In particular, I_l for $l = 2$ and $l = 3$ is equivalent to DkS and DkSH with hyperedges of cardinality* 3*, respectively. When modifying I_2 such that all classifier costs (instead of only singletons) are uniform, the inapproximability of the problem is retained.*

Proof. We next show that I_2 and *DkS* are different formulations of the same problem. The proof for I_3 is completely analogous and thus omitted.

We denote a *DkS* instance by $\langle G, k \rangle$, where G is a graph with nodes V and edges E. The bijection between the *DkS* and I_2 instances is as follows: the node-set V corresponds to the properties P (the set of singleton classifiers is thus also V); the edges E correspond to the queries Q; the budget B corresponds to k; any I_2 solution S consisting of a set of classifiers corresponds to the same *DkS* solution S of vertices. The equivalence then follows from observing that the set of edges in the subgraph, G_S, induced by S is the same as the set of queries that are covered by the classifier set S. Since all utilities equal 1, the utility of a classifier set S is the same as the number of edges in G_S.

It remains to prove the final argument in Theorem 3.3, regarding the modified I_2 special case, where all classifiers are feasible and of uniform cost. This is done by showing that any *BCC* algorithm for this modified special case can be adapted such that it only selects singleton classifiers, without reducing the objective value of the solution. Thus, this special case cannot be easier to approximate than I_2, as it would imply an improved algorithm for I_2 and its equivalent *DkS* formulation as well.

The proof combines the following two observations. First, for hard *DkS* instances, where we assume in particular that the optimal solution does not include a component that is a tree (*DkS* has a

simple exact algorithm over trees [44]), it is straightforward to produce a solution where the number of edges in the subgraph is at least k. Concretely, any non-tree connected component of size k has this property (if all connected components are smaller than k, one can select a union of the densest connected components). The second observation is that every classifier of length two covers exactly one query (the one with the same property), and contributes to covering no other query. From these two observations it follows that no solutions of k classifiers of length 2 can exceed a utility of k, that is a lower bound on the trivial solution of k singleton classifiers corresponding to the k vertices in the *DkS* context as described above. More generally, assume that a solution in the *BCC* context is a union $S = S_1 \cup S_2$, where S_i consist only of classifiers of length i. Then one can modify the solution S, by replacing S_2 with $|S_2|$ singleton classifiers that correspond to a non-tree connected subgraph in the corresponding *DkS* instance, that contains at least $|S_2|$ edges. This results in a solution of at least the same utility, thereby completing the proof. □

4 ALGORITHM

In this section, we devise a *BCC* algorithm that, despite the inapproximability bounds (Section 3), is demonstrated to perform well over real-world data (Section 6). Before presenting our solution, we note that it has been observed in [23] that in real-life workloads most queries are of length at most 2. This fact was exploited there to design an algorithm (for the non-budgeted problem) that first solves the problem over the subset of queries where $l = 2$, and then extends the solution to the residual queries. Our solution exploits this property as well, and, accordingly, we first describe an improved algorithm for $BCC_{l=2}$, based on a reduction to *DkS*, before presenting its extension for the general case.

Importantly, while we leverage the Set Cover solution of [23] (which is unrelated to *DkS*) in a black-box manner as a heuristic local search optimization, all other components of our algorithm are entirely different from the methods of [23], as our generalized problem is likely much harder to approximate. In particular, we must leverage an effective heuristic for the *DkS* special case, and, thus, apply a more granular treatment in our reduction, compared to the Set Cover setting, to ensure that the performance of our algorithm for the general *BCC* problem roughly preserves the performance ratio of the *DkS* heuristic.

4.1 Algorithm for $l = 2$

The first phase of our algorithm for $BCC_{l=2}$ breaks down the problem, such that we need to solve a Knapsack and a *QK* (Definition 2.6) instance, with the optimal solution to one of these problems yielding at least half of the optimal utility of the original *BCC* instance. If most of the utility can be derived from the Knapsack instance, then we can provide an effective solution that yields at least half of the optimal utility for the *BCC* instance, as the Knapsack problem can be approximated to arbitrary precision (Theorem 2.3). However, when the utility derived from the Knapsack instance is insufficient, we need to solve the much harder *QK* problem, which is the focus of all the subsequent phases of the algorithm. Specifically, we will first show that we can modify the currently best approximation algorithm for *QK* [62] to derive improved worst-case guarantees for *QK* and $BCC_{l=2}$. However, since this algorithm is still not sufficiently scalable and is tailored mostly to the worst-case instances, we further modify several of its key components to use the state-of-the-art *HkS* (Definition 2.6) heuristic [41] and prove that our new reduction (to *HkS*) makes better use of this heuristic in terms of the approximation factor.

To describe how to break down the $BCC_{l=2}$ problem into the Knapsack and *QK* subproblems, we need the following definitions and observations, which we also illustrate with examples.

BCC(i) subproblems. Given a query q, we call a set S_i of i classifiers that cover q an *i-cover* if any proper subset of S_i does not cover q. We accordingly denote by $BCC(i)$, for $i \leq l$, the modified *BCC* problem where for each given solution (classifier set) S, a query q is covered by S if and only if S contains an i-cover of q. To illustrate, consider the following example.

Example 4.1. In a (standard) *BCC* instance, where the query set is $Q = \{xyz, xy, x\}$ (for brevity, we omit here the input costs, utilities, and budget bound, as these are inconsequential for the arguments in this example) if we select the classifier set $S = \{X, XY, Z\}$, then all three queries are covered. However, in the $BCC(1)$ instance over the same input, the classifier set S only covers the queries x and xy, as these are the only queries for which S contains a 1-cover. Namely, x is 1-covered by X, and xy is 1-covered by XY (in general, in $BCC(1)$, a query can only be covered by the identical classifier). Similarly, for $BCC(2)$ over the same input, S covers only xyz (with the two classifiers XY and Z). In contrast, the singleton query x cannot be 2-covered by any classifier set, and the query xy can only be 2-covered by $\{X, Y\}$ (note that the selection $\{X, XY\}$ is not a 2-cover of xy since X is dispensable). Lastly, in the corresponding $BCC(3)$ instance, no query is 3-covered. Moreover, only xyz can be 3-covered (and only via the set $\{X, Y, Z\}$).

Each query covered by an optimal *BCC* solution is i-covered for at least one $i \in [l]$ (recall that l is the maximum length of any query in the input). Thus, at least $1/l$ of the optimal utility is derived by at least one of these cover types.

Observation 4.2. *Given a BCC input with an optimal solution of utility U_O, at least one of the problems $BCC(i)$ over the same input, has a solution of utility at least U_O/l.*

It follows that we can partition *BCC* into the $BCC(i)$ subproblems, solve each separately, and choose the best solution, such that the overall approximation factor is higher by at most l than the worst factor of any subproblem.

Knapsack subproblem. For $BCC(1)$, each query can only be covered by the classifier that is identical to it, which implies the following extension of Theorem 3.1 (Section 3).

Observation 4.3. *$BCC(1)$ is equivalent to the Knapsack problem.*

QK subproblem. In $BCC_{l=2}(2)$, only queries of length 2 and singleton classifiers are relevant. Moreover, each query xy can only be 2-covered by the set $\{X, Y\}$. This implies a straightforward equivalence to *QK*.

We next formalize this observation and then illustrate it with an example.

Observation 4.4. *$BCC_{l=2}(2)$ is equivalent to QK when modeling it as a graph, where the nodes are the classifiers, the edges are the*

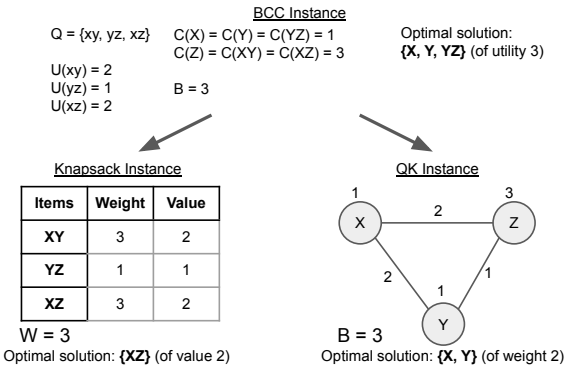


Figure 2: Example of a *BCC* instance with $l = 2$, separated into Knapsack and *QK* instances. The optimal solution of each instance is depicted next to the corresponding input.

queries (i.e. a query xy is an edge connecting X and Y), the node costs and edge weights are the costs and utilities, respectively, and the budget B is the same.

The following example illustrates the above approach.

Example 4.5. Consider the $BCC_{l=2}$ input depicted on the upper half of Figure 2. Its partition into a Knapsack instance (modeling the $BCC(1)$ subproblem), and a *QK* instance (modeling the $BCC(2)$ subproblem), is depicted on the bottom of the figure. Next to each of the three inputs, the corresponding optimal solution is presented.

Observe that, w.r.t. the optimal solution of the *BCC* instance, the query yz is 1-covered by YZ yielding utility 1, and the query xy is 2-covered by $\{X, Y\}$ yielding utility 2. Correspondingly, the solution YZ also yields value 1 in the Knapsack instance, and the solution $\{X, Y\}$ also yields weight 2 in the *QK* instance. This demonstrates that the optimal utility is partitioned across the two subproblems. Moreover, in the Knapsack instance, YZ is not even the optimal solution, as XZ is more valuable. This demonstrates that the worst-case factor of 2 is not necessarily lost in the performance ratio when partitioning the *BCC* instance. In this case, the solutions to the Knapsack and *QK* instances both provide utility 2 in the original *BCC* context, and choosing any of them results in a $(2/3)$-approximation.

Algorithm with worst-case bounds. As explained above, our algorithm separates $BCC_{l=2}$ into a $BCC(1)$ subproblem, that can be approximated to arbitrary precision via a Knapsack algorithm (Theorem 2.3), and a $BCC(2)$ subproblem that can be solved via a $\tilde{O}(n^{0.4})$ PTIME algorithm [62] (our code solves both problems in parallel). Selecting the best of the two solutions guarantees $\tilde{O}(n^{0.4})$-approximation. Moreover, we show that a modification of the algorithm in [62], denoted henceforth as A_T^{QK}, improves this factor to $\tilde{O}(n^{1/3})$, which carries over directly to $BCC_{l=2}$.

Lemma 4.6. *There are $\tilde{O}(n^{1/3})$-approximation algorithms for QK and $BCC_{l=2}$.*

Proof. We next outline, for completeness, the $\tilde{O}(n^{0.4})$-approximation algorithm for *QK* devised in [62], and then explain the simple modification that improves its performance to $\tilde{O}(n^{1/3})$. We remark that, for simplicity, our description slightly differs from the algorithm in [62] (we focus on the worst-case instances, as we only modify the operation of the algorithm over these inputs), however, all differences are inconsequential for the performance analysis.

Normalization. The first step is to normalize the edge weights and node costs. For this, we divide all edge weights by w_{max}/n^2, where w_{max} is the maximum weight of any edge that can be covered (we can assume w.l.o.g. that all edges can be covered with the given budget, as otherwise edges with overly-expensive endpoints can be pruned). This ensures that all edge weights are of order $O(poly(n))$. We then remove all edges of weights below 1. Since there are less than $n^2/2$ such edges, the sum of weights of all these edges is less than half of the weight of the edge of the highest weight, and therefore, by discarding these edges we lose at most a factor of 2 in the performance ratio. Lastly, we round all edge weights down to the nearest power of 2. This also loses at most a factor of 2 in the optimality.

We perform a similar operation on the costs: we divide the budget and the node costs by B/n. It is shown in [62], that we can assume that all node costs are at most $B/4$ (as a solution can contain at most 4 such nodes, and we can solve the problem separately for each such choice, out of the $O(n^4)$ possibilities, retaining a PTIME performance). It is further shown in [62] that when reducing the budget by a constant factor the loss in the optimal weight is of a $O(1)$ factor. Hence, we will in the sequel perform several such operations, without accounting for the optimality loss which is bounded by a constant factor. We can, thus, select all nodes of cost at most 1, as this uses up at most half of the budget. Additionally, we round down the budget to the nearest power of 2 and similarly round up the node costs to the nearest power of 2 (the loss in optimality is smaller than in the case where we instead decrease the budget by a factor of 4, which, as mentioned above, has a negligible effect). This overall ensures that the remaining node costs are all within the range of $[1, B/4]$.

Partitioning the edges into $O(\log^3 n)$ subgraphs. Following the normalization, we examine separately $O(\log^3 n)$ subgraphs of the input graph G, such that the edges are partitioned across all these subgraphs. We then solve the *QK* instance over each subgraph separately and choose the solution of the highest weight. Since the union of the edges in all instances is the original edge set, it follows that the optimal weight that can be covered in at least one of the instances is at least a $1/O(\log^3 n)$-fraction of the optimal weight that can be derived in the original graph. Therefore, by solving the problem separately over each subgraph we lose at most a factor of $O(\log^3 n)$ in the performance, which is negligible compared to the final loss of a $O(n^{1/3})$ factor. Observe that without the normalization, the lost factor would be $O(\log^2 B \cdot \log w_{max})$, which may be arbitrarily larger.

Specifically, we next examine separately all subgraphs of G, indexed by $\{i, j, t\}$, where $i, j, t \in [O(\log n)]$ and $i \geq j$, such that subgraph $G_{i,j,t}$ contains exactly the edges of weight 2^t that connect a node of weight 2^i to a node of weight 2^j (the node set in each subgraph is the union of all endpoints of these edges).

When solving the QK problem over each graph, $G_{i,j,t}$, note that we can ignore the edge weights, as these all have the same value of 2^t. Therefore, the objective simplifies to finding a subgraph that maximizes the *number* of the edges.

Furthermore, note that if $i = j$, then all node costs are uniform as well, which means that the budget constraint simplifies into a cardinality bound of selecting at most $k = B/2^i$ nodes. Hence, we can apply the $\tilde{O}(n^{1/4})$-approximation DKS algorithm of [7] over any graph $G_{i,i,t}$, ensuring a performance ratio of $\tilde{O}(n^{1/4})$ for the QK problem as well.

We next handle the harder case where $i > j$.

Solving QK on a simplified bipartite graph. Observe that when $i > j$, the graph is bipartite since every edge connects a node of weight 2^i and a node of weight 2^j. Moreover, we can normalize the costs to derive a simpler cost scheme, by dividing the budget and all costs by 2^j. Therefore, we get the following bipartite graph: the node set is $L \cup R$, where all nodes in L are of cost 1 and all nodes in R are of weight $w \geq 2$, and all edges are in $L \times R$. Also note that following the costs normalization in the first step, we have an upper bound of order $O(n)$ on w.

Over this graph, we run one of the following two procedures, P_1, and P_2.

In P_1, we select the $B/2w$ (we ignore negligible rounding issues) nodes in R of the highest degree (recall that we can ignore the edge weights), denoting this set by R', and then select the $B/2$ nodes in L of the highest degree into R' (i.e. the highest degree in the graph were we only retain R' from R), denoting this set by L'. The output is $L' \cup R'$. It is shown that the approximation factor of this procedure is $O(n/B)$.

The operation of P_2 is slightly more involved. Specifically, each node in R is replaced by w copies of cost 1 that are connected to the same nodes in L as the original node. Over this graph we run the $\tilde{O}(n^{1/4})$-approximation DKS algorithm of [7]. Let L'' denote the subset of L selected by this algorithm, and let B'' denote the remainder of the budget after the selection of L''. We next select the B''/w nodes of R (in the original graph, and not the blown-up graph with the copies) of the highest degree into L''. The final output is $L'' \cup R''$. Since the DKS algorithm was employed over a graph whose size may reach $O(nw)$, the overall approximation factor of P_2 is $\tilde{O}((nw)^{1/4})$. Moreover, since $w \leq B$, the approximation factor may be presented more loosely as $\tilde{O}((nB)^{1/4})$.

The last step of the QK algorithm is as follows: if $B \geq n^{0.6}$, then we run P_1 (and return its output), and otherwise we run P_2. This results in an overall approximation factor of $O(n^{0.4})$, which occurs in the worst case where $B = 0.6$ (observe that higher budget bounds improve the performance of P_1 and lower budget bounds improve the performance of P_2).

Our modification. Finally, we describe our modification.

First consider the following procedure, denoted by P_3, that can be employed over the bipartite graph above (over which we employed P_1 and P_2): return the node $v*$ of the highest degree in R, and all its neighbors in L (or, more precisely, as many as possible without exceeding the budget). We next analyze the performance of this procedure. Consider the optimal solution O, and let R_O and L_O denote the nodes selected into O from R and L, respectively (i.e. $O = R_O \cup L_O$). Let k_R denote the cardinality of R_O. Observe that the budget constraint implies that $k_R = O(B/w)$. Further note that for at least one node $v_R \in R_O$ its degree into L_O is at least a $1/k_R$-fraction of the overall number of edges in O. Hence, the solution $v_R \cup L_O$ is a (k_R)-approximation. Lastly, since the degree of $v*$ is at least the degree of v_R into L_O, then the solution returned by P_3 is also a (k_R)-approximation of the optimal solution, which can be reformulated as a $O(B/w)$-approximation.

The overall modified algorithm over the bipartite graph is as follows: we run all three procedures P_1, P_2 and P_3, and select the best of the three solutions. This ensures an approximation factor of

$$O(\min\{n/B, (nw)^{1/4}, B/w\}).$$

A simple analysis shows that the worst-case parameter choices that maximize the above expression are $B = n^{2/3}$ and $w = n^{1/3}$. For this choice, all three subexpressions equal $O(n^{1/3})$ (note that any other parameter choice improves at least one of the subexpressions), which is the overall approximation factor of our modified algorithm (and thus we get a $\tilde{O}(n^{1/3})$ performance ratio over the original QK instance as well). □

As noted above, the A_T^{QK} algorithm is, however, impractical (and hence also the corresponding BCC algorithm), due to scalability issues and its guarantees being of a $poly(n)$ order, corresponding to worst-case instances, which may not resemble real-world data. Nevertheless, we show below that we can modify key components of A_T^{QK}, to derive a more practical alternative, denoted by A_H^{QK}, that is both scalable and tailored to real-world performance. We will use in A_H^{QK} the idea from A_T^{QK} of replacing each node with multiple copies (described in the proof of Lemma 4.6), however, we do so in a more general graph setting, and thus use a more involved procedure to transform the DkS solution over this graph into a QK solution over the original graph.

Heuristic QK algorithm. The first modification we perform in A_T^{QK} is replacing the worst-case-oriented DkS algorithm of [7] with the HkS heuristic in [41], that has been shown to produce solutions close to optimal on large graphs. This allows to significantly improve both the efficiency and the quality of the solution. However, even assuming an optimal solution to the DkS instance, the corresponding BCC solution may yield only a $O(\log^3 n)$-fraction of the optimal utility, (improved only to $O(\log^2 n)$, if we more generally reduce to HkS instead of DkS). We, therefore, as a second modification, devise a different reduction, such that this polylogarithmic factor is reduced to a much smaller constant.

We will prove the following worst-case bound on the performance ratio of our algorithm, which is conditioned on the performance of the HkS algorithm (that was shown in [41] to provide solutions that typically exceed 65% − 80% of the optimal value).

Theorem 4.7. *Given an HkS algorithm with performance ratio $\alpha = O(1)$, the performance ratio of A_H^{QK} is at most $(5\alpha + \epsilon)$ (for any $\epsilon > 0$), implying a $(7\alpha + \epsilon)$-approximation algorithm for $BCC_{l=2}$.*

Importantly, we will show in our analysis that most of the loss in optimality pertains to degenerate cases. This is corroborated by our experiments (Section 6), where the eventual value derived by the QK algorithm always exceeds that of the HkS algorithm.

We next provide the description of the A_H^{QK} algorithm interleaved with the proof of Theorem 4.7.

Preprocessing. We first explain how we transform the input for the *BCC* problem, such that in the corresponding *QK* instance all costs are integers in the range $[1, B/2)$.

First, we can select all classifiers of cost 0 (i.e. add them to the solution). Following this selection, the input is transformed as follows. Any query that becomes covered is removed from the input, with its utility counted towards the objective function. Any query for which no relevant classifier is selected remains as it is. The only remaining case corresponds to a query of the form xy, where the classifier X is selected and Y is not. Given the X is already selected, we can consider, in the residual problem, the selection of Y as a 1-cover. Therefore, we remove the query xy from the $BCC(2)$ instance, and modify the Knapsack instance for the $BCC(1)$ subproblem as follows: we add the item Y to the input (alternatively, it may already exist if the *BCC* query set contains the query y), such that its weight is the cost of the classifier Y (as is the case for all other items), and its value is the sum of the utilities of the queries of the form XY where X is selected (and we also add the utility of the query y, if such exists).

This transformation ensures that the covering possibility of selecting both X and Y remains in the solution space, except that it is now transferred to the solution space of $BCC(1)$. However, this leads to some performance loss: since the classifier XY is also an item in the Knapsack instance, if we select both XY and Y, then we would count the utility of the query xy twice. It follows that we lose a factor of 2 in the approximation guarantee of $BCC(1)$, which now becomes $(2+\epsilon)$.

Overall, once we prove that the approximation factor for the $BCC(2)$ instance is $(5\alpha+\epsilon)$ (where $\alpha = O(1)$ is the assumed approximation factor of the *HkS* algorithm as specified in Theorem 4.7), the overall approximation factor for the *BCC* instance is derived as follows. Let β denote the fraction of the optimal utility of the *BCC* instance that can be derived in the $BCC(1)$ instance. The corresponding fraction of the optimal utility for the $BCC(2)$ instance is therefore at least $(1-\beta)$. Temporarily ignoring the ϵ factor for simplicity, we have that the 2-approximation of the $BCC(1)$ instance provides at least a $\beta/2$-fraction of the optimal utility, whereas the (5α)-approximation of $BCC(2)$ provides a $(1-\beta)/(5\alpha)$-fraction of the optimal utility. Taking the maximum of both expressions, simple algebra implies that the worst-case corresponds to the case where $\beta = 2/(2+5\alpha)$. For this β value, both solutions provide at least a $(1/(2+5\alpha))$-fraction of the optimal utility. Finally, since $\alpha \leq 1$, we get the worst-case factor of $(7\alpha+\epsilon)$, as stated in the Theorem 4.7.

Continuing with the description of the preprocessing phase, we can also prune all classifiers whose cost exceeds B. When removing such a classifier X of length 1, we can remove from the input the query x, if it exists, and also remove any query xy of length 2 that contains x from the $BCC(2)$ input (xy can only be covered by XY which is captured by the $BCC(1)$ instance). Similarly, when pruning a classifier XY of length 2, we simply remove the item XY from the $BCC(1)$ instance.

We next explain how preprocessing the *QK* input (this is done over the $BCC(2)$ instance before reducing it to *QK*, but explaining the procedure is more natural in the context of the *QK* input) allows to remove all nodes (recall that the nodes correspond to the singleton classifiers) whose cost is in $[B/2, B]$. We call such nodes *expensive nodes*.

Observe that the number of expensive nodes in the optimal solution is at most 2. We can, thus, for each distinct selection of at most 2 expensive nodes solve the residual problem, where all such nodes are removed, and choose the best solution of all selections. Specifically, if there are exactly two expensive nodes, then the solution contains no other nodes, and we can iterate over all $O(n^2)$ selections of two expensive nodes (without computing any corresponding residual *QK* instance). To account for the remaining possibilities, we solve the problem over the instance where all expensive nodes (and the incident edges) are removed (which corresponds to the case where the optimal solution contains no expensive nodes), and similarly, for each choice of a single expensive node, deduct it from the budget, and solve the residual *QK* problem with the reduced budget over the input where once again all expensive nodes are pruned. Note that if the number of expensive nodes is n_b, then we must solve (n_b+1) *QK* instances. If n_b is large then solving many instances is compensated by the fact that each residual instance contains only $(n-n_b)$ nodes and a reduced budget, which helps improve the running time (additionally, in practice, there are many expensive nodes only for very small budgets, which correspond to instances that can be solved much faster). As for computing the residual instance after the selection/removal of these nodes, we use the same procedures as described above for nodes of cost 0 or cost above B.

Lastly, we can assume that all remaining costs are integers, as otherwise, we can round up each cost to the nearest multiple of some $\epsilon \in (0,1)$, and then multiply all costs (and the budget) by $1/\epsilon$. This rounding loses at most an ϵ factor in the approximation.

Random bipartite graph. Given as input a budget B and a graph $G=(V,E)$, preprocessed as described above, with node costs given by $c(\cdot)$ and edge weights given by $w(\cdot,\cdot)$, the first step is to transform G into a bipartite graph. To do so, we adopt a randomized procedure pointed out in [53], in the context of a spectral *DkS* algorithm. To ensure that the probabilistic error on its performance guarantee is sufficiently small ($O(1/n)$), we employ it $\log n$ times (we do so in parallel) each time running the entire algorithm and eventually choosing the best solution of all iterations. In each iteration, in the first step of the random procedure, we partition V into two sets $\bar{L}$ and $\bar{R}$, assigning each node independently to one of the sets with uniform probability. We then derive from G a bipartite graph $\bar{G}=(V,\bar{E})$ where $\bar{E}=E\cap(\bar{L}\times\bar{R})$, with the same costs and weights. With probability exceeding $(1-1/n)$, in at least one iteration, the value of the optimal *QK* solution in $\bar{G}=(V,\bar{E})$ exceeds half of the optimal value in G. Therefore, the randomized procedure adds a factor of 2 to the worst-case performance ratio.

Solving HkS on a blown-up graph. The next step is to eliminate costs, so that an *HkS* algorithm can be employed. Specifically, a graph $\hat{G}=(\hat{V},\hat{E})$ is derived from $\bar{G}$, as follows. Each node $v\in V$ is replaced in $\hat{V}$ by $c(v)$ copies, where $c(v)$ is the cost of v. For every edge $\{v,u\}\in\bar{E}$ there are $c(v)\cdot c(u)$ edges in $\hat{E}$ connecting all copies of v to all copies of u, where the weight of each such edge is $\frac{w(v,u)}{c(v)\cdot c(u)}$. The sets of copies of $\bar{L}$ and $\bar{R}$ are denoted in $\hat{G}$ by $\hat{L}$ and $\hat{R}$, respectively. We then run an *HkS* algorithm over $\hat{G}$ with $k=B/2$, yielding a solution $\hat{S}$ (since the other half of the budget may be used

later when transforming $\hat{S}$ into a solution over the G, we allocated only half of the budget for the *HkS* algorithm, which, as shown in [62], loses at most a factor of 2 in the optimality.) Given $\hat{S}$, we further use $L = \hat{L} \cap \hat{S}$ and $R = \hat{R} \cap \hat{S}$, to denote the subset of the solution in each of the two sets of the partition.

For any edge $\{v, u\} \in \bar{E}$, the sum of weights of the edges between the copies of v and u is $w(v, u)$. It follows that every *QK* solution in $\bar{G}$ has a corresponding solution in $\hat{G}$, with the same cost and weight, where all the copies of the first solution are selected. Therefore, the weight of the optimal solution in $\hat{G}$ can only exceed the optimal weight in $\bar{G}$. Hence, if we translate the *HkS* output $\hat{S}$ back into a solution over $\bar{G}$ with the same weight (and at most twice the cost), then the performance ratio would be at least as good as the performance of the *HkS* algorithm.

Swapping copies. We call a node in $\bar{G}$ *partially selected* if the solution over $\hat{G}$ (which at this point of the algorithm is $\hat{S}$) contains some but not all of its copies. Similarly, we call a node in $\bar{G}$ *completely selected* if the solution over $\hat{G}$ contains all of its copies.

We next describe a procedure where we swap in the solution copies of one node in $\bar{G}$ for another, to reduce the number of partially selected nodes to at most 2. We do this while ensuring that the total weight of the subgraph induced by $\hat{S}$ never decreases. We first perform this procedure on L and then on R (or, more precisely, first over $\hat{L}$ and then over $\hat{R}$).

Given R, for each node in $\hat{L}$, we refer to the sum of the weights of the edges incident to it with endpoints in R as its *weighted degree*. The swapping procedure consists of the following two phases:

(1) First, we swap a copy selected in L with a non-selected copy of a different node whose weighted degree (into R) is higher (i.e. remove the former copy from L and add instead the latter). This is repeated until there are no two nodes whose copies can be swapped based on this rule. Once this phase is completed, all copies in L of partially selected nodes are of the same weighted degree, which is upper bounded by the weighted degrees of all the copies of the completely selected nodes.
(2) Next, fix some arbitrary order over the partially selected nodes and swap a selected copy of a node of a lower position with a non-selected copy of a node of a higher position. This operation is also repeated until there are no two nodes whose copies can be swapped by this rule.

Observe that after the above swapping procedure is completed, there is at most one partially selected node in $\hat{L}$, as at least one of the two phases above would swap copies of any two partially selected nodes. Moreover, note that we have never swapped a copy of a higher weighted degree with a copy of a lower weighted degree (also observe that all copies of the same node have the same weighted degree). Therefore, the weight of the subgraph induced by $L \cup R$ has either increased or remained the same.

We next perform the same swapping procedure over $\hat{R}$ w.r.t. the new L set. It follows from the same arguments that the weight of the subgraph has not decreased and that at most one node is partially selected in $\hat{R}$.

Final selection. Up to this point, we have lost in the worst-case performance ratio of the *QK* algorithm a factor of 2 when transforming the into graph into a bipartite graph, another factor of 2 when using a reduced budget for the *HkS* algorithm, and a factor of α due to the suboptimal approximation guarantee of the *HkS* algorithm. We next describe the final step of the algorithm, which contributes another factor of $(5/4+\epsilon)$ to the loss in optimality, and thus we get the overall worst-case performance ratio of $(5\alpha+\epsilon)$, as stated in Theorem 4.7.

Let V_L and V_R denote the subsets of V (the node set of G and $\bar{G}$) whose copies are completely selected in L and R, respectively.

First, note that if all nodes in G whose copies are selected in $L \cup R$ are completely selected, then we can simply produce the Solution V_L and V_R. As explained above the subgraph of $\bar{G}$ induced by $V_L \cup V_R$ (assuming all selected nodes are *completely* selected in $\hat{S}$) is of the same weight as the subgraph induced by $L \cup R$ in $\hat{G}$ and also of the same cost (the sum of the costs of all the copies of a node is the cost of the node). Therefore, as G has the same costs as in $\bar{G}$ and contains the edges of $\bar{G}$ along with other edges, the corresponding subgraph in G induced by $V_L \cup V_R$ is also of the same cost and at least the same weight. Therefore, the performance ratio over $\hat{G}$ carries over to G by producing all the nodes that are completely selected.

Otherwise, if exactly one node in V is partially selected, we can use the half of the budget we set aside (recall that we used only $B/2$ for the *HkS* algorithm) to select all the remaining copies of this node. This is feasible since our preprocessing step ensured that no node costs more than half the budget. This only improves the weight of the solution, and this improvement carries over to the final solution $V_L \cup V_R$.

The only remaining case is when there are two partially selected nodes in V: a node $u_L \in \hat{L}$ and a node $u_R \in \hat{R}$. If the remaining half of the budget is sufficient to add all the remaining copies of both nodes, then we do so, and once again solve the problem as in the previous cases. If, however, the budget is insufficient, then we examine separately two possible cases.

(1) Case *I*: if the sum of the weights of all the edges connecting the selected copies of u_L and u_R does not exceed $1/5$ of the overall weight of $\hat{S}$, then we remove from $\hat{G}$ all edges connecting copies of u_L and u_R. Note that the remaining weight of the subgraph induced by $\hat{S}$ is a $(4/5)$-fraction of the weight before the removal of the edges. Next, assume w.l.o.g. that the weighted degree of each copy of u_L is at least that of each copy of u_R. Then we can swap all selected copies of u_R with non-selected copies of u_L until there are no more selected copies of u_R. Observe that this can only increase the weight of the solution, and does not change the cost, and moreover all nodes are now completely selected, which implies the same solution as before of outputting all the completely selected nodes in $\hat{L} \cup \hat{R}$.
Note that it is impossible that we select all copies of u_L before de-selecting all copies of u_R, as this would imply that we have only one partially selected node, for which the budget is necessarily sufficient, contradicting the assumption that we cannot add all copies of both nodes to the solution.
Overall, as stated before, we lost a factor of $5/4$ in the performance ratio in this case.
(2) case *II*: if the sum of weights of all the edges connecting the selected copies of u_L and u_R does exceed $1/5$ of the overall

weight of $\hat{S}$, then we modify L and R to contain only the full sets of copies of u_L and u_R, respectively, and once again return V_L and V_R (which in this case is simply $u_L \cup u_R$), as the final output over G. Note that since no node costs more than half the budget, we can always afford a solution that contains two nodes.

We next show that this solution also loses at most a 5/4 factor in the performance ratio compared to the *HkS* solution over $\hat{G}$, and thus concluding the proof of Theorem 4.7.

Concretely, if half of the budget was insufficient to add all the remaining copies of u_L and u_R to the solution $\hat{S}$, then the overall number of such non-selected copies exceeds $B/2$. Moreover, since neither u_L nor u_R cost more than $B/2$, it follows that if the fraction of the selected copies of u_L out of all the copies of u_L is β, then the corresponding fraction of selected copies of u_R is at most $(1-\beta)$. Therefore, the weight of the edge connecting u_L and u_R is $\frac{1}{\beta(1-\beta)}$ times larger than the sum of edge weight of the selected copies of u_L and u_R in $L \cup R$. This expression always exceeds 4 (and equals 4 exactly for $\beta = 1/2$). Since the sum of weights between the selected copies of u_L and u_R is 1/5 of the overall weight produced by the solution $L \cup R$, we get an overall factor of 5/4, which completes the proof.

Finally, observe that the hardest cases where we lose most of the optimality pertain to inputs where the solution contains a node of high cost so that we cannot select all its copies without using a large fraction of the budget, and of very high weighted degree (the weights of the edges incident to it in the produced solution represent a constant fraction of the overall weight of the optimal solution) so that we cannot simply ignore it with a negligible effect on the performance ratio. In practice, such degenerate cases are rare, as only for very small budgets we may have a single classifier that consumes most of the budget, and also must be selected to ensure high utility.

We also remark that over practical inputs, if we can disregard the two nodes that are partially selected, then instead of performing the swapping procedure, we can simplify the remainder of the algorithm (following the employment of the *HkS* algorithm) to solve two knapsack instances over $\bar{L}$ and $\bar{R}$. We, however, do not provide here the details of this simplification.

4.2 Algorithm for l > 2

We are now ready to present our heuristic algorithm in its most general form, which also covers the case of $l > 2$. For simplicity, we focus in our description on the case of $BCC_{l=3}$, however, our arguments also apply to larger l values, analogously.

We next overview the high-level ideas, which we also illustrate with an example, before describing the exact steps taken by the algorithm.

Overview. We first observe that, similarly to Theorem 4.7, for $l > 2$, our solution of the subproblems $BCC(1)$ and $BCC(2)$ still guarantees an approximation factor of $O(1)$, albeit of a larger constant, assuming the performance ratio of the *HkS* solver is also $O(1)$. While $BCC(1)$ is the Knapsack problem for any l, in $BCC(2)$ for queries of length over 2 the problem becomes more complex, as there are multiple overlapping 2-covers. For example, there are 6 different 2-covers of a query of length 3. Hence, when modeling $BCC(2)_{l=3}$ as a *QK* instance, where the nodes are the classifiers and an edge connects any two classifiers that form a 2-cover, the objective value of any solution may be up to 6 times larger than its utility in the *BCC* context, as the same query may be covered multiple times (i.e. 6 different edges in the *QK* input represent the same *BCC* query). This 6 factor, however, can then be reduced, as such worst cases imply a redundancy in the selected classifiers. Specifically, given a *QK* output S that covers the query set Q_S, finding a set of classifiers of the lowest cost that covers Q_S is exactly the *MC*3 problem (Definition 2.4), for which we can use the algorithm from [23] (Theorem 2.5). The fraction of the budget saved by the solution to the *MC*3 instance compared to S can then be used for the residual problem (i.e. to cover the remaining uncovered queries, given the classifiers selected so far). A second important observation is that when solving the residual problem, new covering possibilities may be considered in $BCC(1)$ and $BCC(2)$, as illustrated by the following example.

Example 4.8. Consider the input query set $Q = \{xyz, xyw\}$ (as in Example 4.1, we omit here the input costs, utilities, and budget bound, since our high-level arguments are not based on concrete numerical computations). When solving the corresponding $BCC(1)$ and $BCC(2)$ instances, the only two covering possibilities not included in the combined solution space of these two problems is the cover of xyz by $\{X, Y, Z\}$ and the cover of xyw by $\{X, Y, W\}$, as these are 3-covers. Assume, next, that the solution for the $BCC(2)$ instance produced by our A_H^{QK} algorithm is $\{YZ, XZ, Y\}$, and that this solution was chosen over the Knapsack Solution for the $BCC(1)$ instance. Observe that only the xyz query is covered. However, the solution contains a redundancy as the sets $\{YZ, XZ\}$ and $\{Y, XZ\}$ are both 2-covers of the same query. If we then run an *MC*3 algorithm that searches for the lowest-cost set of classifiers that covers xyz, it may output the less costly solution $\{XZ, Y\}$, which saves the cost of YZ that can be instead used to select other classifiers. We note that since the *MC*3 problem is *NP*-hard (Theorem 2.5), the *MC*3 algorithm is not guaranteed to improve on the previous solution (which in the above case was $\{YZ, XZ, Y\}$), even if there indeed exists a less costly solution that covers the same queries, and if this is the case, then we retain the previous solution instead of the *MC*3 output. Therefore, the *MC*3 algorithm in our context is essentially a local search optimization. Also note that, while the *MC*3 algorithm is oblivious to the budget bound, if, nevertheless, the *MC*3 solution does improve on the previous solution, then the newer (*MC*3) solution is necessarily within the budget constraint, as the costlier solution also did not exceed the bound.

Next, given that we have so far selected $\{XZ, Y\}$, only the query xyw remains uncovered in the residual problem. Moreover, as Y is already selected, it is not necessary to select in the residual problem classifiers that contain y, and thus a cover of the xw component in xyw is sufficient. This implies that we can treat both $\{XYW\}$ and $\{XW\}$ as a 1-cover of xyw in the residual problem (however, selecting both, would once again imply a redundancy, that can be ameliorated with an *MC*3 algorithm). Note that XYW is a 1-cover of xyw in both the original and the residual problems, however, XW is only a 1-cover in the residual problem, since it must be paired with the Y classifier selected earlier. Similarly, the 2-covers of xyw are

now $\{X, W\}$, $\{XY, W\}$, $\{X, WY\}$, and $\{XY, WY\}$. Note that, there are no longer 3 covers of xyw, as these require the selection of Y, which is already selected and does not appear in the residual problem. Therefore, all covering possibilities are now considered (albeit with some redundancies).

Lastly, observe that the above arguments for the simplification of the residual problem also apply for queries whose length exceeds 3. For instance, selecting the classifier XY implies that all covering possibilities in the residual problem for the query $xyzw$ are either 1-covers or 2-covers.

We, therefore, initially use only half of the budget to solve the $BCC(1)$ and $BCC(2)$ subproblems (via our algorithm for $BCC_{l=2}$), to guarantee that a sufficient fraction of the budget is available for the residual problem, that now contains a larger fraction of the complete solution space of *BCC*.

To ensure, however, that the solution space does not become too large and hinders scalability, we use two preprocessing procedures to prune the input classifier set.

We next list the steps of the algorithm outlined above, as depicted schematically at high-level in Algorithm 1.

Algorithm 1: A^{BCC} (high-level scheme)

1 preprocessing: apply two pruning methods to reduce the number of input classifiers
2 allocate half of the budget to solve the $BCC(1)$ an and $BCC(2)$ subproblems via the algorithm for $BCC_{l=2}$ (Subsection 4.1)
3 test whether the solution produced in the previous step can be improved cost-wise via the *MC*3 algorithm in [23]
4 while the budget allows covering more queries repeat the following steps:
5 compute the input for the residual problem
6 perform the two steps in lines 2 and 3, using the remainder of the budget (instead of only half of it, as before)

Preprocessing (line 1 in Algorithm 1). For $l > 2$, the number of relevant classifiers for $BCC(2)$ may be significantly larger than for $l = 2$. Therefore, as a preliminary step, we prune classifiers from the solution, using two procedures with a bound on the incurred error. The first procedure removes every classifier of length $r > 1$ that can be replaced by several shorter classifiers whose total cost is at most r times its cost. For example, if the classifiers XYZ, X, Y, and Z all cost 1, then we can remove XYZ since any solution that uses it can instead use X, Y and Z such that the set of covered queries can only increase. In particular, this means that for instances with uniform costs, the solution space is reduced to using only singleton classifiers. Note, however, that in some edge cases, where the budget is very small (relative to the costs of the classifiers), this pruning rule above is not applied. Concretely, for any given query, if the pruning rule would retain only short classifiers, such that any combination of these classifiers that covers the query exceeds the budget (i.e., this query can no longer be covered), then we do not prune the longer classifiers relevant to this query (one such query is sufficient to "protect" the longer classifier from pruning). The second pruning procedure is based on *weighted leverage scores* [10, 46, 53], which are derived via spectral methods over the adjacency matrix of the *QK* input graph.

Solving BCC(1) and BCC(2) subproblems (line 2). We use half the budget to solve each of the subproblems, $BCC(1)$ (via the Knapsack algorithm) and $BCC(2)$ (via our algorithm for $l = 2$), and select the solution S of the highest utility of the two solutions.

Improvement via MC3 algorithm (line 3). Let Q_S denote the set of queries covered by S. We next employ the *MC*3 algorithm from [23], over the input consisting of Q_S and all the classifiers from the *BCC* input that are relevant for covering Q_S. We denote the output of the *MC*3 algorithm by S'.

Solving iteratively residual problems (lines 4 – 6). We next compute the residual problem, given the selection of S', and use the remainder of the budget (instead of only half the budget, as before) to employ over it the algorithm for $BCC(1)$ and $BCC(2)$, along with the *MC*3 optimization, as we did in the previous two steps. This is repeated iteratively until the budget is consumed entirely (i.e. the *MC*3 algorithm no longer produces a less costly solution). The selected set of classifiers at this point is the final output.

5 COMPLEMENTARY OBJECTIVES

In the two previous sections, we presented hardness bounds and an algorithm for *BCC*. In this section, we examine alternative objectives that may be of interest in practical scenarios where there is some flexibility in the budget constraint. For example, companies may aim to find the classifier set of minimum cost that reaches a given utility target (this is a direct generalization of [23]) or to find a classifier set that maximizes the ratio of utility to cost. We show that our *DkS*-based analysis methods for *BCC* can also be applied to derive complexity bounds and algorithms for these two problems, defined formally below.

Definition 5.1. In the *Generalized MC*3 *problem* (*GMC*3) the input is $\langle Q, U, C, T\rangle$, where Q, U and C, as in *BCC*, are the queries, utilities, and costs, respectively, with $T \in \mathbb{R}_+$ representing a target utility value. The goal is to find a set of classifiers of minimum cost, yielding utility at least T.

Definition 5.2. In the *Effective Classifier Construction problem* (*ECC*), the input is $\langle Q, U, C\rangle$, where Q, U and C, are as in *BCC*, and the goal is to find a classifier set that maximizes the ratio of its utility to its cost.

We first provide hardness bounds and an algorithm for *GMC*3, showing that despite it containing *MC*3 (Definition 2.4) as a special case (where the target utility is the sum of utilities of all queries), the problem is more similar theoretically to *BCC*. We then examine *ECC* and prove that it is much easier than *BCC* and *GMC*3, as it admits an exact PTIME solution for $l = 2$, and a constant approximation for general (constant) l.

The GMC3 problem. Since *GMC*3 is complementary to *BCC* in the sense that the roles of the objective and the constraint are reversed, it is natural to transform our *DkS*-based analysis, to use the analogously complementary problem to *DkS*, the *Smallest p-Edge Subgraph problem* (*SpES*), where the goal is to find a subgraph of the minimum number of nodes that contains at least p edges. The best known approximation factor for *SpES* is $\tilde{O}(n^{0.17})$, and the best known approximation factor for its hypergraph extension, *SpESH*, where all hyperedges are of cardinality 3, is $\tilde{O}(n^{0.62})$ [6].

Several results link potential improvements in the approximations of *DkS* and *SpES* [15], and, in particular, the "Dense vs Random" conjecture [17, 48] implies for both problems that the best

known approximation guarantees are tight. An adaptation of our methods to using *SpES* instead of *DkS* implies the following results.

Theorem 5.3. *The GMC3 problem is NP-hard, and for $l = 2$ or $l \geq 3$ is at least as hard as SpES or SpESH, respectively. However, given an α-approximation BCC algorithm, one can derive an $O(\alpha \log n)$-approximation GMC3 algorithm with a $O(\frac{1}{n})$ additive error in T.*

Proof. We start with proving the hardness results. To this end, consider the special case of *GMC3* where all costs of singleton classifiers equal 1, all costs of other classifiers equal ∞, all utilities equal 1, and all queries are of length exactly l. Over such inputs, the objective simplifies to finding the smallest set of classifiers that covers at least T queries. This special case is equivalent to *SpESH* where all hyperedges are of cardinality l (for $l = 2$ this is exactly *SpES*), which implies the hardness. The equivalence follows the same transformation as in the case of the equivalence of the special case of *BCC* and *DkSH* (Theorem 3.3). Concretely, we model every singleton classifier as a vertex, and every query is a hyperedge (i.e. the query xyz is the hyperedge consisting of the three vertices x, y, and z), and the utility target T becomes the lower bound p on the number of hyperedges. The objective of *GMC3* then becomes selecting the smallest set of nodes that contain at least $p = T$ hyperedges, which is exactly the *SpESH* problem.

We next prove that given an α-approximation algorithm for *BCC*, we can derive an $O(\alpha \log T)$-approximation algorithm for *GMC3*. The proof is a fortuitously simple generalization of a proof shown in [15] for an analogous relationship between *DkSH* and *SpESH*.

Given the input $\langle Q, U, C, T\rangle$ for *GMC3*, let B denote the cost of the corresponding optimal solution. Let $A(Q, U, C, B)$ denote the output of the α-approximation algorithm for *BCC*, A, employed over the input $\langle Q, U, C, B\rangle$. We can assume w.l.o.g. that B is known, as we check all possible values up to a negligible error of epsilon, and solve the problem for each guess (while this is in *PTIME*, due to the normalization process of the weights described in the proof of Lemma 4.6, a better solution in practice would be using a binary search, for which it is easy to show that a constant loss in the budget implies at most a constant loss in the optimality).

We first rescale the utilities and T (this phase does not exist in the less general algorithm in [15]), similarly to the rescaling in the proof of Lemma 4.6: we divide T and all utility values by $\frac{u_{max}}{n^{l+1}}$, where u_{max} is the maximum utility of any query that can be covered within the budget B. This ensure that $T = O(poly(n))$, and thus $\log(T) = O(\log(n))$. We further remove all queries whose utility is less than 1. Since the sum of utilities of all such queries is less than $O(u_{max}/n)$ (there are at most $O(n^l)$ queries), and thus also less than $O(T/n)$ (we assume w.l.o.g. that $u_{max} \leq T$, otherwise we can find the optimal solution by covering only the query of utility u_{max}). It follows that the removal of these queries adds at most an $O(1/n)$ additive error in the utility target T, as specified in the Theorem, which we henceforth ignore.

We then have the following algorithm for *GMC3* (which, as mentioned above, is a straightforward generalization of the algorithm in [15]), that initializes $Q' = \emptyset$ and repeats the following steps until the sum of utilities of the queries in Q' exceeds T:

(1) Let $S' = A(Q, U, C, B)$, and let Q'' denote the queries covered by S'.
(2) Let $Q' \leftarrow Q' \cup Q''$.
(3) Remove Q'' from Q (but keep all classifiers).

We next analyze the approximation factor of this algorithm. This analysis is also entirely analogous to the analysis in [15]. We, nevertheless, provide it for completeness.

Suppose that at iteration i we added classifiers of total cost c_i and that before the start of the iteration we had already added to the solution (Q') queries of total utility $T - t_i$. This implies that the subset of the queries in the optimal solution that has not yet been covered is of total utility at least t_i. Therefore, there is a set of classifiers of cost bounded by B (the same set which is the optimal solution in the original instance) that covers these uncovered queries. It then follows that the utility of the queries added in iteration i was at least t_i/α. Therefore, the utility we still cover after iteration i is $t_{i+1} \leq t_i - t_i/\alpha = t_i(1 - 1/\alpha)$. Hence, by induction, after j iterations, the utility we still need to cover is bounded by

$$t_{j+1} \leq T(1 - 1/\alpha)^j \leq Te^{\frac{j}{\alpha}}.$$

Therefore, after at most $\alpha \ln T$ iterations, the total utility covered is at least T (since the remaining utility in the above inequality is less than 1, however, we only retained queries of utility at least 1). Moreover, our normalization ensured that $\alpha \ln T = O(\alpha \log n)$. Finally, since each iteration adds classifiers of cost at most B, we have a total cost of at most $O(B\alpha \log n)$, proving the theorem. □

As mentioned above, *GMC3* generalizes *MC3*. We note, however, that since *SpES* and *SpESH* are special cases of *GMC3*, the Set Cover solution of [23] (for *MC3*) is no longer relevant for *GMC3*.

The ECC problem. For *ECC* we can again apply the approach of reducing it to a density problem. As *ECC* maximizes the ratio between the objective and the constraint of *BCC*, an analogous transformation of the *DkS* problem yields the Densest Subgraph problem (*DS*). In *DS* the input is a graph with weights on both nodes and edges, and the goal is to find a subgraph such that the ratio of the sum of edge weights to the sum of node weights is maximized. This problem, even on hypergraphs, can be solved exactly in PTIME [35]. Reductions to *DS*, mostly analogous to our *DkS* reductions, imply the following results.

Theorem 5.4. *The ECC problem can be solved exactly for $l = 2$, and admits $O(1)$-approximation for $l > 2$.*

Proof. We first show a reduction from *ECC* with $l = 2$ to *DS* on graphs, and then generalize it to the case of $l > 2$.

Given an *ECC* input $\langle Q, U, C\rangle$ with $l = 2$, we construct an input graph G for *DS* as follows. Each singleton classifier is a vertex whose weight is the cost of the classifier, and each query xy of length 2 connects the vertices X and Y, and its weight is the utility of the query. We also add a special vertex $v*$ of weight 0 and for every query x of length 1 we add the edge $(X, v*)$ whose weight is the utility of x.

Let S denote the solution of the exact *DS* algorithm [35] over G, and let r denote the ratio of utility to cost achieved by S. Furthermore, let S' denote the classifier of length 2 that has the highest ratio of the utility of the query it covers (if there is a classifier XY, then there must be a query xy) and its cost, and let r' denote this ratio. If $r' \geq r$, we output S' and otherwise we output S.

We next show that this algorithm produces the optimal *ECC* solution. First, observe that if we restrict the solution space of *ECC* to singleton classifiers, then the problem becomes equivalent to solving *DS* over G. This is because the sum of costs of a classifier set S'' is exactly the sum of weights of the corresponding vertex set $S'' \cup v*$, and the sum of utilities of the queries covered by S'' is exactly the sum of weights of the edges in the subgraph induced by $S'' \cup v*$ (recall that the weight of $v*$ is zero). Thus, if the optimal solution to the *ECC* instance consists of only singleton classifiers, then r is necessarily the optimal ratio.

Similarly, note that if we restrict the solution space of *ECC* to solely classifiers of length 2, then the optimal ratio is necessarily r', since every classifier of length 2 covers exactly one query, and every query is covered by only one such classifier. Thus, including in the solution other classifiers, except for S', can only decrease the ratio.

Finally, if the optimal solution of *ECC* is $S_1 \cup S_2$, where S_i is a set of classifiers of length i, then since neither the ratio of S_1 nor the ratio of S_2 are greater than $\max\{r, r'\}$, the overall ratio of $S_1 \cup S_2$ is at most $\max\{r, r'\}$ (since no query requires a combination of classifiers from both S_1 and S_2 to be covered), which proves the theorem for $l = 2$.

For the case of $l > 2$, we generalize the construction of the graph G to the construction of the hypergraph H, which serves as the input for *DS*. Specifically, each classifier of length at most $l - 1$ is a vertex in H (its weight as before is the cost of the classifier), and each subset of these classifiers that covers a query (and that any proper subset of this subset does not cover it - i.e. it is a minimal cover) is a hyperedge whose weight is the utility of the query (for queries of length 1, we once again, to avoid self-loops, add a vertex $v*$ of cost 0). We then compute two solutions: the *DS* solution over H, and the classifier of length l of the maximum ratio between the utility of the query it covers and its cost. Finally, we output the solution of the highest ratio of the two solutions.

The proof that an optimal solution can consist of either only classifier of length l or only classifiers of length at most $l-1$, is analogous to the proof for the case of $l = 2$. However, the only difference here is that for classifiers of length at most $l - 1$, the *DS* solution is no longer the exact *ECC* solution but a $O(1)$-approximation of it. This follows from the fact that each query has multiple corresponding hyperedges in H, and thus we are overcounting if multiple such hyperedges are contained in the subgraph of the solution. This multiplicity, however, has a constant upper bound. For example, for $l = 3$ there are at most 7 hyperedges that represent the same query (i.e., for the query xyz there are 7 combinations of classifiers that cover it, not including the classifier XYZ or covers where there is a redundant classifier). More broadly, since $l = O(1)$ the maximum number of such hyperedges is constant for any l, which proves the theorem. □

6 EXPERIMENTAL STUDY

In this section, we present the experimental evaluation of our *BCC* algorithm performed over various datasets, including real-world data of *BCC* use-cases, followed by an evaluation of our proposed algorithms for the *GMC*3 and *ECC* problems (Section 5). We start with describing our experimental setup for the *BCC* problem, then present the evaluation results for *BCC*, and conclude with describing our experimental setup and results for the complementary problems.

6.1 Experimental Setup

Our algorithms were implemented using Python, and we ran the experiments on a server with 128GB RAM and 32 cores. In addition, we used external implementations of the *HkS* algorithm of [41] and the *MC*3 algorithm of [23]. To evaluate our solution, we performed a set of extensive experiments on a publicly available real-world dataset, a larger private dataset provided by a large e-commerce company[2], including costs and utilities provided by the company's business analysts, and a synthetically generated dataset.

Datasets. As noted above, we performed our evaluation over three datasets. For all datasets, we tested a wide range of budget bounds, detailed in our presentation of the experimental results.

- **BestBuy (BB)** - First, we used a small, publicly available, dataset from BestBuy, which had been used by [18, 23] for their evaluation. The dataset consists of roughly 1000 queries with 725 distinct properties from the electronics domain. The average query length is 1.4, with more than 95% of the queries containing at most 2 properties, and 65% of the queries being of length exactly 1, which is the most common query length in all datasets. This dataset includes the number of times each query was searched, and we also use this number as the utility score (based on the logic that popular queries are more important to compute correctly). No classifier costs, however, are included, and, hence, we assume uniform costs, as discussed in Section 2.
- **Private (P)** - The second dataset is private and comes from a large e-commerce company. It consists of $5K$ popular queries (with $2K$ distinct properties) of various lengths (1 to 5 properties), utilities, and classifier costs. This dataset is a union of several sub-datasets pertaining to different categories of products (mostly Electronics, Fashion, and Home & Garden). These queries are taken from the search logs of Q1 2021 and represent the actual complete query set marked by analysts as top priority for result set improvement. The average length of a query is 1.7. Concretely, more than 95% of the queries are of length at most 2, and 55% are of length exactly 1. The costs represent a scaling by a factor of N (an internal measure of the e-commerce company) of the estimated monetary cost of training each classifier. After the normalization, the costs are in the range $[0, 50]$, with the average cost being roughly 8. The classifiers whose cost is ∞, which, as explained in Section 2, correspond to classifiers whose construction is deemed impractical, are omitted from the input. These cost estimations were determined by business analysts based on the estimated number of training examples domain experts must label to train the corresponding classifier to the required precision. Similarly, the utility score of each query is derived by the analysts as a combination of the importance of the corresponding product category (i.e. based on how important it is considered for the company to improve query results in this category) and the search frequency of

[2]Company name omitted due to privacy considerations.

the query. As explained in Section 2, the units of measure of the utility are inconsequential for our model, and thus these scores can be rescaled by any factor. For simplicity, we scale these into the range $[1, 50]$.

- **Synthetic (S)** - The third dataset is generated synthetically, to consist of $100K$ queries. We note that to facilitate the scalability tests, the size of this query set greatly exceeds typical query load sizes targeted by classifier construction (as reported by analysts). The costs and utilities are integers drawn independently from a uniform distribution over the ranges $[0, 50]$ and $[1, 50]$, respectively. The length of any generated query equals i with a probability $\frac{1}{2^i}$, i.e. half of the queries are of length one, a quarter of the queries are of length two, and so on. This captures the inverse correlation of query frequency and length that exists in practice (and, in particular, in the two previous datasets). Queries generated with a length exceeding 6 are omitted because companies do not allocate resources for such rare queries [27]. The average query length resulting from this process is 1.8. Into each query, we select uniformly properties from a pool of $10K$ properties. This dataset is regenerated for each separate experiment.

Algorithms. We compare the following four *BCC* algorithms. Since no competing algorithm exists in the literature, we examine natural greedy and random baselines. In particular, the *IG*2 algorithm described below is an adaptation to our context of the greedy Set Cover algorithm used to solve the *MC*3 problem in [23]. Note that the baselines used for the evaluation of *GMC*3 and *ECC* solutions, are described separately in Subsection 6.3.

- **RAND** - This simple baseline randomly selects in each iteration one of the classifiers whose selection will not exceed the budget.
- **IG1** - An iterative greedy algorithm that in each iteration computes for each uncovered query the least costly set of classifiers that covers it (by checking all $O(1)$ relevant sets), and then selects the classifier set that maximizes the ratio of the utility of the corresponding query and its cost (we only count the costs of the classifiers that have not been selected in the previous iterations).
- **IG2** - Another iterative greedy algorithm, that in each iteration selects a single classifier. Concretely, it computes for each classifier the sum of utilities of the queries that contain it and then selects the classifier that maximizes the ratio between the corresponding sum of utilities and its cost.
- $\boldsymbol{A^{BCC}}$ - Our proposed algorithm (Algorithm 1 in Section 4).

Evaluation outline for *BCC*. We compared the algorithms listed above over our datasets, both in terms of the overall utility of the selected classifier set and the overall running time (we executed the random *RAND* baseline 5 times in each experiment, and averaged the results). We tested each dataset with a wide range of budget values. To compute an upper bound on this range, we solved the *MC*3 problem (Definition 2.4) using the algorithm of [23], as the cost of the produced solution is sufficient to cover all queries. To understand how our algorithm compares with the optimal solution, we also ran experiments on small subsets of the input, where a brute force approach could find the best solution. Lastly, we examined the effect of our preprocessing step (the first step of Algorithm 1) on the execution time and the solution quality.

6.2 BCC Evaluation Results

We next present the results for the *BCC* problem and discuss important insights.

Solution quality. Figures 3a, 3b, and 3c depict the utility achieved by each algorithm over the *BB*, *P*, and *S* datasets respectively, for some of the examined budget values. In all examined cases the A^{BCC} algorithm always achieved the best performance.

Figure 3a depicts the utilities achieved over the *BB* dataset for 4 different budget values. The ranking of the algorithms for all budget bounds is the same: A^{BCC} is the best performing algorithm, followed by *IG*2, *IG*1, and *RAND*. We note that the *BB* dataset is very sparse since each property appears in a very small number of queries, and the corresponding *HkS* instance (Definition 2.6) is, therefore, very sparse as well. Moreover, both A^{BCC} and *IG*2 produce solutions where almost all of the utility comes from covering singleton queries. This allows *IG*2 to achieve results close to A^{BCC}. However, over all other tested datasets, the gap significantly widens as there are solutions containing classifiers that help cover simultaneously queries of different lengths, which are found via good approximation of the *HkS* instance. This is evident over the *P* and *S* datasets, for which results for a selection of budget values are depicted in Figures 3b and 3c, respectively. Here again the ranking of the algorithms is the same, except that *IG*1 outperforms *IG*2 over small budgets. In particular, the gap between A^{BCC} and the second-best algorithm over *P* is much larger than over *S*. This occurs because the probabilistic generative process that constructs *S* results in a, roughly speaking, more "balanced" *HkS* graph, whereas the *P* dataset can be better exploited by the *HkS* algorithm that focuses on a union of low-cost dense subgraphs.

Lastly, Figure 3d depicts the comparison of A^{BCC} and the brute-force algorithm (that also uses pruning) over a subset of the *P* dataset (we tested small query subsets that pertain to very specific subdomains, such as "iPhones" queries). Naturally, there is some loss in optimality compared to the (non-practical) exhaustive search. However, the loss is always less than 20% on these small instances. The brute-force results over small synthetic instances showed roughly the same trends and hence omitted.

Insights. We next present additional findings and insights derived from the quality experiments. We first note the effect of *diminishing returns*, which is particularly noticeable over the *P* dataset: we see that the growth in the utility is not quadratic (as one could expect, e.g., in the extreme case of *DkS*, where the order of the number of edges is quadratic in the subgraph size for very dense graphs), but rather sublinear. This is because most of the utility is typically concentrated in much smaller subsets of the instance. Moreover, the algorithm initially covers the queries of the highest utility where less costly classifiers are available, and as the budget increases, such "easy" query subsets become much rarer.

One important corollary is that compared to the budget required to cover all queries (which is computed in the *MC*3 setting of [23]) the budget required to cover a large fraction of the utility is much smaller. For instance, over the *P* dataset the budget required by the *MC*3 algorithm exceeds 8000, however, a budget of 4000 is sufficient

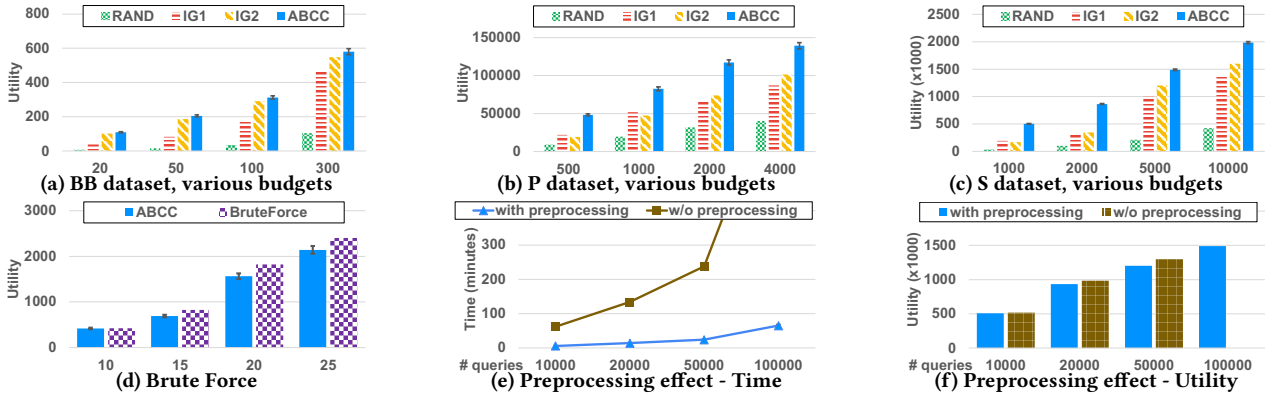


Figure 3: Experimental results for BCC

to cover 75% of the total utility of all queries (which is roughly 186K). We also note that the real quarterly budget provided to us by the analysts for this dataset is roughly 2000, and this is sufficient for 65% of the total utility. Note that in particular this very loose bound on the optimal utility implies that the performance ratio of our algorithm is at most 4/3 for 75% of the budget, and roughly 1.5 (i.e. loss of around 35%) for 50%. Moreover, to provide additional context for the percent loss compared to total utility (which may of course greatly exceed the optimal solution, which we cannot compute) we note that the total utility possible over the *BB* dataset is roughly $1K$, whereas over the S dataset it is roughly $2.5M$.

We also note that for the "real" budget of 2000 over P, roughly 51% of the utility comes from queries of length 2 and about 47% from singleton queries (the rest is over longer queries). Note that this means that queries of length 2 are slightly more covered compared to their incidence in the input, which is due to the fact the A^{BCC} is able to locate dense subgraphs (with relatively more weight from edges compared to nodes), although the solution also contains sparser subgraphs.

Second, we note a characteristic of the real-world datasets (*BB* and even more so P): popular queries (which are queries of high utility) tend to have popular subqueries. For example, if people often look for "black Adidas shoes" then people also often look for "Adidas shoes" and "black shoes". This property is exploited well by our algorithm, A^{BCC}. Specifically, in many cases, the QK solution is better than the Knapsack solution (see Section 4 for details), and consists mostly of singleton classifiers. Therefore, when covering popular queries of length 2 the QK solution also tends to cover many popular queries of length 1 that are captured by the Knapsack instance. A similar phenomenon occurs when we solve the residual problem, as the classifiers used for the shorter popular queries, tend to be relevant for longer queries that contain the former queries as subsets, which simplifies the residual problem (see Algorithm 1).

Scalability and Preprocessing. Finally, we discuss the scalability, and, in particular, the effect of our preprocessing procedure (step 1 in Algorithm 1), which reduces the size of the input at the cost of a provably small loss in the optimality. The relevant experiments involved testing a wide range of budget values (including the budget sufficient to cover all queries), and for each budget bound we generated the synthetic datasets multiple times with different sizes of the input query set. As the general trends were similar over all tested budget values, we only show representative results for a budget bound of 5000. Concretely, figures 3e and 3f depict, respectively, the running time and utility of A^{BCC} with and without preprocessing, over the S dataset, where we varied the number of generated queries from 10K to 1000K and always used the same budget bound of 5000. Note that on instances with over 50K queries the variant of the algorithm without preprocessing did not terminate. One can see that the degradation in performance caused by the preprocessing is negligible, however, the gain in efficiency is significant. In particular, over the dataset with 100K queries A^{BCC} (with preprocessing) produced a solution after 65 minutes, and even on much larger budgets the running time over S did not exceed 80 minutes, which is affordable running time for an offline task.

Preliminary end-to-end results. Lastly, we present some general findings reported to us by our business collaborators (that have provided the P dataset, and are also evaluating our proposed solution) in terms of the accuracy of the cost estimates, and the extent to which the results of a sample of the newly-covered queries have improved. Note that these evaluations and findings are entirely independent of our solution, as the cost estimates are the same for all baselines, and the improvement to the query is the same regardless of the solution that suggested to cover it. Therefore, this is only provided as supplementary material to add some context to the practical setting and demonstrate its validity, and is in no way a direct evaluation of our algorithms (a detailed end-to-end analysis that compares the overall improvement to the search results of our algorithm to other baselines, would require not only a lot of time but also that companies separately adhere to suggestions of inferior algorithms for the sake of the evaluation).

We first report that of a random sample of 20 constructed classifiers, that cost estimates were on average lower than the actual costs by roughly 6%. This underestimation is theoretically equivalent to reducing the overall budget by 6%, and, as we have proven in Section 4, a small constant (multiplicative) modification to the budget results in at worst a slightly larger reduction in the optimal utility. Nevertheless, it would be interesting to investigate techniques that more accurately predict, e.g., the number of necessary training examples necessary to train a given classifier to a given accuracy threshold. Moreover, as classifiers are used only after reaching 95% accuracy on test sets, we report that the original estimates were in almost all cases sufficient to exceed 90%.

Moreover, on a random sample of 20 queries that have been covered by classifiers (due to privacy concerns, we cannot disclose information on the queries or concrete internal evaluation techniques), the size of the complete result sets compared to the previous result set (that has leveraged retrieval methods based on similarity of embeddings as described in the introduction) has in all cases increased by more than 200% (which is not surprising, since these queries are known to be problematic, and hence targeted by the business analysts), and in 3 queries by more than 500%. It has been reported that the precision has also improved, compared to noisier heuristics, however no concrete values were provided to us.

6.3 *GMC*3 and *ECC* Evaluations

While the primary focus of the present work is on the *BCC* problem, we, nevertheless, also present, in this subsection, preliminary evaluation results for our proposed algorithms for *GMC*3 and *ECC* (Section 5). As the main purpose of discussing these complementary problems is to demonstrate that one can approximate them well by adapting our solution methods for *BCC*, we present evaluation results that corroborate this claim by demonstrating the superiority of our proposed solutions over more naive baselines. Nonetheless, an in-depth empirical study of these problems, that includes optimizing the above algorithms with various heuristics and pruning procedures, analogously to our optimization of the A^{BCC} algorithm, is beyond the scope of this paper.

We first describe the baselines used for evaluating our *GMC*3 solution, followed by the evaluation results, and then do the same for *ECC*.

GMC3 baselines. We evaluate the following four *GMC*3 algorithms, that are almost entirely analogous to the four algorithms used for the evaluation of our *BCC* solution. In particular, the first three algorithms listed below are identical to their *BCC* counterparts, except that the stopping condition is reaching the utility target rather than the budget bound.

- **RAND(G)** - This simple baseline randomly and uniformly selects in each iteration one classifier until the target utility is reached or exceeded.
- **IG1(G)** - An iterative greedy algorithm that in each iteration computes for each uncovered query the least costly set of classifiers that covers it (by checking all $O(1)$ relevant sets), and then selects the classifier set that maximizes the ratio of the utility of the corresponding query and its cost (we only count the costs of the classifiers that have not been selected in the previous iterations).
- **IG2(G)** - Another iterative greedy algorithm, that in each iteration selects a single classifier. Concretely, it computes for each classifier the sum of utilities of the queries that contain it and then selects the classifier that maximizes the ratio between the corresponding sum of utilities and its cost.
- $\boldsymbol{A^{GMC3}}$ - Our proposed algorithm for *GMC*3, described in the proof of Theorem 5.3, that iteratively employs our A^{BCC} algorithm over the same input along with an estimated budget bound that fits this algorithm. The naive implementation of this algorithm, as described in Section 5, tests every possible budget bound to find the best value (it selects the best solution out of all examined values). However, to allow for scalable performance, in our implementation of A^{GMC3}, we use instead a binary search for the best budget bound over a reduced relevant range approximately inferred from running the *MC*3 solution of [23], which computes an upper bound on the budget required for reaching the sum of the utilities of all input queries.

GMC3 evaluation results. We now present the evaluation results for the *GMC*3 problem. Figures 4a, 4b, and 4c depict the budget used by each algorithm over the *BB*, *P*, and *S* datasets, respectively, for some of the examined utility targets. In all examined cases the A^{GMC3} algorithm achieved the best performance.

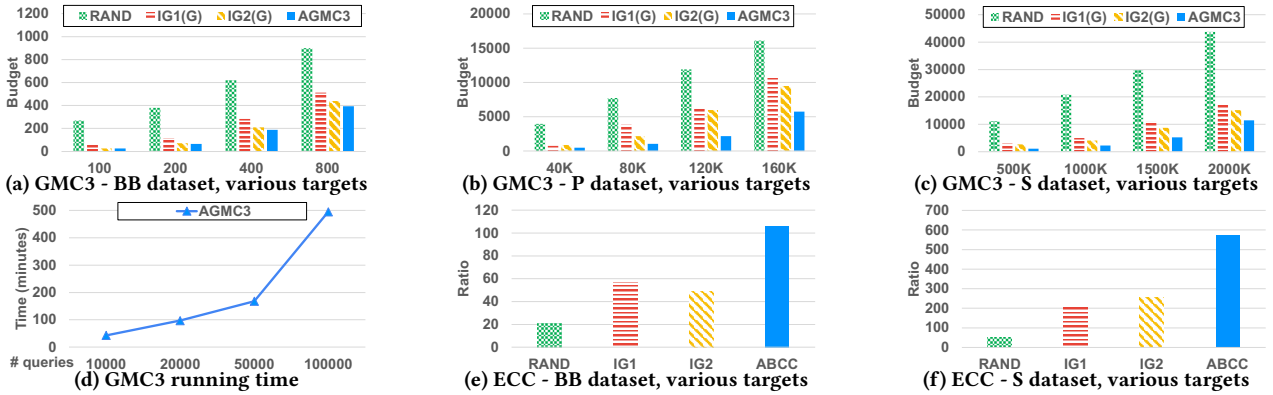
Recall that the A^{GMC3} algorithm essentially consists of running A^{BCC} several times in succession (taking the union of the produced solutions). Hence, since all examined algorithms are almost entirely analogous to the *BCC* algorithms, and due to the inherent relation between the two problem definitions, all quality evaluations demonstrate the same trends as in the evaluation of our *BCC* solutions. One small difference, however, is that the gap between our algorithm and the other baselines is diminished, compared to *BCC*, partly because the binary search heuristic adds some error.

In terms of the scalability of A^{GMC3}, however, its running time is considerably higher the A^{BCC}. This is expected, since many iterations are discarded, and are only used for guiding the binary search for a good budget bound for the input of the underlying A^{BCC} algorithm. Moreover, for the best found budget bound, A^{BCC} is iteratively employed 2 to 4 times (with rare exceptions), until the target utility is reached.

As mentioned, optimizing this algorithm, analogously to our optimization of A^{BCC} is beyond the scope of this work. Nevertheless, the running time is, arguably, at least reasonable over the real-world datasets, and is affordable for an offline task, even over the larger synthetic dataset. The running times over the synthetic dataset (the experimental setup is analogous to the *BCC* evaluation), for a representative utility target of 150K are depicted in Figure 4d.

ECC baselines. We evaluate the following four *ECC* algorithms. These baselines are, once again, almost entirely analogous to the four baselines used for *BCC* and *GMC*3. In particular, besides our proposed algorithm, the three baselines operate the same as in the previous problems, with two modifications: (1) the stopping condition is covering all queries, and (2) the output is not the final set of classifiers, rather the set that achieved the best ratio across all iterations. For completeness, we again define these baselines below.

- **RAND(E)** - This simple baseline randomly and uniformly selects in each iteration one classifier until all queries are covered.
- **IG1(E)** - An iterative greedy algorithm that in each iteration computes for each uncovered query the least costly set of classifiers that covers it (by checking all $O(1)$ relevant sets), and then selects the classifier set that maximizes the ratio of the utility of the corresponding query and its cost (we only count the costs of the classifiers that have not been selected in the previous iterations).
- **IG2(E)** - Another iterative greedy algorithm, that in each iteration selects a single classifier. Concretely, it computes for each classifier the sum of utilities of the queries that



(a) GMC3 - BB dataset, various targets (b) GMC3 - P dataset, various targets (c) GMC3 - S dataset, various targets

(d) GMC3 running time (e) ECC - BB dataset, various targets (f) ECC - S dataset, various targets

Figure 4: Experimental results for GMC3 and ECC

contain it and then selects the classifier that maximizes the ratio between the corresponding sum of utilities and its cost.

- A^{ECC} - Our proposed algorithm for *ECC*, described in the proof of Theorem 5.4. As described in Section 5, this algorithm employs in a black-box manner an algorithm for finding the densest ratio-wise subgraph in weighted hypergraphs. There are several such exact algorithms in [35], along with a faster r-approximation algorithm, where r is the maximum cardinality of a hyperedge in the graph, which in our setting is typically close to 2. As we do not have access to the implementation of the exact algorithms, we have used the simple greedy approximation algorithm.

ECC evaluation results. Unlike the previously examined problems, in *ECC* there is no analogous dimension to the utility target or the budget constraint. For each dataset, there is only one given solution. The best ratios produced by each algorithm over the *P* and *S* datasets, respectively, are shown in Figures 4e and 4f (the trends over *BB* are roughly the same, and are omitted to facilitate a more succinct visual presentation of the results). Note that the best ratio is of a different order of magnitude over each dataset, as, intuitively, the optimal ratio may potentially reach much higher values over large and "dense" instances. We note that the best found ratio over the *P* dataset corresponds to a total cost of roughly 400, whereas the best found ratio over *S* corresponds to a total cost of roughly 900.

The running time of A^{ECC} varies from several seconds to several minutes depending on the size of the dataset. Thus, the solution for this problem is much more scalable compared to *BCC* and *GMC*3. To some extent, this is due to the fact that we used the faster approximation algorithm from [35], instead of one of the more exact algorithms. A more thorough examination of possible underlying algorithms for weighted subgraphs to use in our *ECC* algorithm, along with various optimizations, analogous to our A^{BCC} algorithm, is beyond the scope of this work, as our focus in on the *BCC* problem. Nevertheless, as *ECC* is less similar to *BCC* than *GMC*3, the potential for improvement in solution quality via an in-depth empirical investigation is likely to be greater.

7 RELATED WORK

We start this section by describing previous work on non-budgeted variants of our problem, and, more generally, work on optimizing the cost of classifier construction and minimization of human effort. We then discuss problems that share some similarities with our setting, highlighting important technical distinctions. Lastly, we review additional *HkS* (Definition 2.6) and *DkS* algorithms that can be paired with our reduction scheme (Section 4).

Non-budgeted variants of BCC. The problem of identifying cost-effective classifiers has been introduced in [18, 22, 23]. However, no budget constraints were taken into account, hindering the practical applicability of these solutions. To address this, we extend the above model with a budget constraint, and, since in our case not all queries are necessarily covered, we also differentiate between queries via utility scores that model how valuable it is for a solution to cover each given query. Due to *BCC* generalizing *DkS*, the set-cover-based methods used for *MC*3 [23] are no longer relevant for our model, and novel techniques needed to be developed.

Economic classifier construction. In recent years, companies have been relying on classifiers for an ever-increasing number of applications, including spam detection [26, 36], identifying helpful sentences from user reviews [21, 34], fraud detection [5, 37, 49], finding a proper category for an item in the company's taxonomy [32, 64, 70], and classifying search queries [11], which is also the focus of our work. As mentioned in the introduction, classifier construction is known to be expensive [67, 68], primarily due to the training process requiring volumes of high-quality labeled training data [40]. In particular, labeling is performed by humans for each data item separately, leading to bottleneck concerns, especially when expertise is required, as experts' time is more valuable [60]. Therefore, much research has been devoted to optimizing the cost of the training process [20, 45]. These works typically focus on the use-case where the properties the classifier must test are given, and the goal is to select the most cost-effective construction methods [54] or devise techniques to minimize the number of data examples required by the method of choice to reach a satisfactory level of precision [14]. Our paper is complementary to these lines of research: given cost estimations for the classifiers, our algorithm identifies the (conjunctions of) item properties.

Importance of accurate meta-data in e-commerce. Maintaining accurate and high-quality metadata is one the key challenges of large e-commerce platforms. In addition to previously mentioned paper [60] by Walmart, that aims to improve product classification accuracy, there are other works from other e-commerce companies, e.g., [51, 66] by Amazon that study the importance of high-quality attributes information. Other companies such as Alibaba [1] and eBay [2] explicitly ask the sellers to provide accurate information about the product and specifically product attributes as part of their guidelines. It is clear that having the most accurate attributes of the products is crucial for e-commerce companies (e.g., for having better search results [28]). Hence the aforementioned and other large e-commerce companies both request the best available data (attributes) from the sellers while uploading new products and, in parallel, train high-quality classifiers. Those classifiers are used to extract the attributes from other provided inputs (e.g., title, image and description). However, to the best of our knowledge our work is the first to address the problem of which classifiers to train in a cost-efficient manner (in terms of required labeled data) under given budget constraint. Specifically, while in our work the goal is to minimize the effort (amount of training data) needed to train classifiers that cover a specific need (in our case - search queries), other works focus on how develop best possible classifiers and generally do not deal with the budget constraint. Hence, our work is complementary to these efforts.

Minimization of crowd work. More broadly, our research falls under the category of works aiming to minimize the effort or involvement of human workers in various tasks that support supervised machine learning [69], e.g., feature selection [52], learning semantic attributes [63], and image tagging [61]. In these tasks, the human component tends to be the costliest [60], and in our problem, as well, the construction costs capture the required human effort.

Related problems. While we are not aware of any work directly comparable to ours, we briefly discuss below three problems that share some similarities with our model.

First, we mention the *Materialized View Selection* problem (*MVS*) [42, 50, 56], where the goal is to materialize, in the context of data warehouses, a set of views (relations) that strike the right balance between optimizing the execution time of answering an expected query workload and minimizing the overall space used. At high-level, *MVS* is somewhat analogous to *BCC*, with views corresponding to classifiers, space - to costs, and execution time - to utility. Even so, the *MVS* problem has much higher inapproximability bounds [38], and we are not aware of any theoretical results or heuristic solution methods that resemble ours. Typical technical modeling distinctions include the fact that each query is covered by only one view (the smallest view that contains its result set) [30], and that the execution time of each query is counted towards the objective function regardless of the view selection [24]. In contrast, in *BCC* queries are either covered entirely (and precisely) or not at all, and the objective measures the utility gained only from covered queries, with no extra penalty for not covering the remaining queries. Correspondingly, the greedy heuristics typically used for *MVS* [24, 25] are unrelated to *DkS*, and thus also do not apply to *BCC*, which generalizes *DkS* (to our knowledge, all top *DkS* heuristics are based on convex optimization [8, 41, 59] and spectral methods [53]). Nevertheless, it may be interesting to explore whether applying our model to the *MVS* setting, allows capturing practical objectives, e.g., storing a set of relations within a space budget, to maximize the utility gain over queries computed by join/union operations, which correspond to the logical conjunction of classifiers in *BCC*.

An even more similar line of research is *Multi-Task Learning* (*MTL*) [55, 71, 72]. In *MTL* there is a set of tasks, and one must select which set of classifiers to construct, such that various combinations of subsets of these classifiers may address these tasks optimally. The relation to our model is clear, as classifiers have more or less the same role, the tasks correspond to queries (there is also typically one primary high-level task which corresponds in our setting to improving query results), and different combinations of classifiers may address (cover) different tasks (queries). Most *MTL* works focus on network architectures and on deriving the possible combinations of classifiers that are most relevant to each task, however, to our knowledge, there is no study of which combinations are the most cost-effective, despite the fact the the implementation of the *MTL* solution also requires human effort in training the classifiers, and is subject to budget limitations. Thus, an interesting future work direction is applying our methods to *MTL*.

A related line of research examines the *Minimum Substring Cover problem* [12, 33], where one searches for a set of strings, such that subsets of these can be concatenated to derive any string in another input set of strings. This problem, however, is much easier to approximate than *BCC*, primarily due to the technical differences between (string) concatenation and logical conjunction (of classifiers). Moreover, the requirement of the solution to cover all input strings is more similar to *MC*3 [23] than to the present work, where we cover only a subset of the queries.

HkS and DkS algorithms. Our *BCC* algorithm leverages the state-of-the-art *HkS* heuristic [41], based on convex optimization. Since our solution is modular and uses *HkS* as a black-box, one can, in principle, use any other *HkS* algorithm. For instance, the algorithms in [43] and [57] were also shown to perform well in practice. Moreover, for instances with uniform utility values, *HkS* simplifies into *DkS* for which there are many additional algorithms to consider. A recent example of an empirically-tested heuristic is [8], and one can also consider superpolynomial algorithms [9].

8 CONCLUSION AND FUTURE WORK

In this work, we study the *BCC* problem of selecting a classifier set of the highest utility in practical settings where a budget constraint is imposed. We showed that *BCC* is as hard as the *DkS* problem and its hypergraph extensions, for which reasonable worst-case guarantees have eluded researchers for decades. Nevertheless, we proposed a practical algorithm that leverages recently-devised *DkS* heuristics and showed experimentally over real-world and synthetic data that it greatly exceeds the worst-case bounds. As explained in Section 5, our methods are also applicable to other problem variants where there is some flexibility in the available budget.

One interesting direction for future work is identifying special cases that allow providing better worst-case bounds. Another intriguing avenue of exploration is generalizing our model to account for utility in partial covers of queries or generalizing the cost function to capture overlaps in classifier construction (see discussion in Section 2).

REFERENCES

[1] Alibaba's Rules for Filling of Product Information. https://rule.alibaba.com/rule/detail/11000682.htm.

[2] eBay's Guidelines for Listing Specifics. https://pages.ebay.com/seller-center/listing-and-marketing/item-specifics.html.

[3] N. Alon, S. Arora, R. Manokaran, D. Moshkovitz, and O. Weinstein. Inapproximability of densest κ-subgraph from average case hardness. *Unpublished*, 1, 2011.

[4] B. Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. *SIAM Journal on Computing*, 42(5):2008–2037, 2013.

[5] J. O. Awoyemi, A. O. Adetunmbi, and S. A. Oluwadare. Credit card fraud detection using machine learning techniques: A comparative analysis. In *ICCNI*, pages 1–9. IEEE, 2017.

[6] A. Bhangale, R. Gandhi, and G. Kortsarz. Improved approximation algorithm for the dense-3-subhypergraph problem. *arXiv preprint arXiv:1704.08620*, 2017.

[7] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an o (n 1/4) approximation for densest k-subgraph. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 201–210, 2010.

[8] P. Bombina and B. Ames. Convex optimization for the densest subgraph and densest submatrix problems. In *SN Operations Research Forum*, volume 1, pages 1–24. Springer, 2020.

[9] N. Bourgeois, A. Giannakos, G. Lucarelli, I. Milis, and V. T. Paschos. Exact and superpolynomial approximation algorithms for the densest k-subgraph problem. *European Journal of Operational Research*, 262(3):894–903, 2017.

[10] C. Boutsidis, M. W. Mahoney, and P. Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 968–977. SIAM, 2009.

[11] A. Z. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang. Robust classification of rare queries using web knowledge. In *SIGIR*, pages 231–238, 2007.

[12] S. Canzar, T. Marschall, S. Rahmann, and C. Schwiegelshohn. Solving the minimum string cover problem. In *ALENEX*, pages 75–83, 2012.

[13] M. Charikar, M. Hajiaghayi, and H. Karloff. Improved approximation algorithms for label cover problems. *Algorithmica*, 61(1):190–206, 2011.

[14] Y. Chen, L. Cheng, and Y. Zhang. Building a training dataset for classification under a cost limitation. *Electron. Libr.*, 39(1):77–96, 2021.

[15] E. Chlamtác, M. Dinitz, C. Konrad, G. Kortsarz, and G. Rabanca. The densest k-subhypergraph problem. *SIAM Journal on Discrete Mathematics*, 32(2):1458–1477, 2018.

[16] E. Chlamtac, M. Dinitz, and R. Krauthgamer. Everywhere-sparse spanners via dense subgraphs. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 758–767. IEEE, 2012.

[17] E. Chlamtáč, M. Dinitz, and Y. Makarychev. Minimizing the union: Tight approximations for small set bipartite vertex expansion. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 881–899. SIAM, 2017.

[18] E. Dushkin, S. Gershtein, T. Milo, and S. Novgorodov. Query driven data labeling with experts: Why pay twice? In *EDBT*, 2019.

[19] U. Feige, M. Seltser, et al. *On the densest k-subgraph problem.* Citeseer, 1997.

[20] G. Forman and I. Cohen. Learning from little: Comparison of classifiers given little training. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 161–172. Springer, 2004.

[21] I. Gamzu, H. Gonen, G. Kutiel, R. Levy, and E. Agichtein. Identifying helpful sentences in product reviews. In *NAACL-HLT 2021, Online, June 6-11, 2021*, pages 678–691, 2021.

[22] S. Gershtein, T. Milo, G. Morami, and S. Novgorodov. Mc3: A system for minimization of classifier construction cost. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2725–2728, 2020.

[23] S. Gershtein, T. Milo, G. Morami, and S. Novgorodov. Minimization of classifier construction cost for search queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1351–1365, 2020.

[24] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for olap. In *Proceedings 13th International Conference on Data Engineering*, pages 208–219. IEEE, 1997.

[25] H. Gupta and I. S. Mumick. Selection of views to materialize under a maintenance cost constraint. In *International Conference on Database Theory*, pages 453–470. Springer, 1999.

[26] V. Gupta, A. Mehta, A. Goel, U. Dixit, and A. C. Pandey. Spam detection using ensemble learning. In *Harmony search and nature inspired optimization algorithms*, pages 661–668. Springer, 2019.

[27] I. Guy. Searching by talking: Analysis of voice queries on mobile web search. In *SIGIR 2016*, pages 35–44, 2016.

[28] I. Guy, T. Milo, S. Novgorodov, and B. Youngmann. Improving constrained search results by data melioration. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1667–1678. IEEE, 2021.

[29] M. T. Hajiaghayi and K. Jain. The prize-collecting generalized steiner tree problem via a new approach of primal-dual schema. In *SODA*, volume 6, pages 631–640. Citeseer, 2006.

[30] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD Record*, volume 25, pages 205–216, 1996.

[31] A. Hassan, X. Shi, N. Craswell, and B. Ramsey. Beyond clicks: query reformulation as a predictor of search satisfaction. In *CIKM*, pages 2019–2028, 2013.

[32] I. Hasson, S. Novgorodov, G. Fuchs, and Y. Acriche. Category recognition in e-commerce using sequence-to-sequence hierarchical classification. In *Proceedings of WSDM*, pages 902–905, 2021.

[33] D. Hermelin, D. Rawitz, R. Rizzi, and S. Vialette. The minimum substring cover problem. *Information and Computation*, 206(11):1303–1312, 2008.

[34] S. Hirsch, S. Novgorodov, I. Guy, and A. Nus. Generating tips from product reviews. In *WSDM*, pages 310–318, 2021.

[35] S. Hu, X. Wu, and T. H. Chan. Maintaining densest subsets efficiently in evolving hypergraphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 929–938, 2017.

[36] A. J. Ibrahim, M. M. Siraj, and M. M. Din. Ensemble classifiers for spam review detection. In *2017 IEEE Conference on Application, Information and Network Security (AINS)*, pages 130–134. IEEE, 2017.

[37] J. Jurgovsky, M. Granitzer, K. Ziegler, S. Calabretto, P.-E. Portier, L. He-Guelton, and O. Caelen. Sequence classification for credit-card fraud detection. *Expert Systems with Applications*, 100:234–245, 2018.

[38] H. Karloff and M. Mihail. On the complexity of the view-selection problem. In *PODS*, pages 167–173, 1999.

[39] Y. Khanna and A. Louis. Planted models for the densest k-subgraph problem. *arXiv preprint arXiv:2004.13978*, 2020.

[40] Y. Ko and J. Seo. Text classification from unlabeled documents with bootstrapping and feature projection techniques. *Inf. Process. Manag.*, 45(1):70–83, 2009.

[41] A. Konar and N. D. Sidiropoulos. Exploring the subgraph density-size trade-off via the lovaśz extension. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 743–751, 2021.

[42] Y. Kotidis and N. Roussopoulos. A case for dynamic view management. *TODS*, 26(4):388–423, 2001.

[43] M. Letsios, O. D. Balalau, M. Danisch, E. Orsini, and M. Sozio. Finding heaviest k-subgraphs and events in social media. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 113–120. IEEE, 2016.

[44] M. Liazi, I. Milis, and V. Zissimopoulos. Polynomial variants of the densest/heaviest k-subgraph problem. In *Proceedings of the 20th British Combinatorial Conference, Durham*, 2005.

[45] N. Lu, G. Niu, A. K. Menon, and M. Sugiyama. On the minimal supervision for training any binary classifier from only unlabeled data, 2019.

[46] M. W. Mahoney and P. Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

[47] P. Manurangsi. Almost-polynomial ratio eth-hardness of approximating densest k-subgraph. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 954–961, 2017.

[48] P. Manurangsi and D. Moshkovitz. Improved approximation algorithms for projection games. *Algorithmica*, 77(2):555–594, 2017.

[49] T. Milo, S. Novgorodov, and W. Tan. Interactive rule refinement for fraud detection. In *EDBT*, pages 265–276, 2018.

[50] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham. Materialized view selection and maintenance using multi-query optimization. *ACM SIGMOD Record*, 30(2):307–318, 2001.

[51] F. Moraes, J. Yang, R. Zhang, and V. Murdock. The role of attributes in product quality comparisons. In *Proceedings of the 2020 Conference on Human Information Interaction and Retrieval*, pages 253–262, 2020.

[52] B. Nushi, A. Singla, A. Krause, and D. Kossmann. Learning and feature selection under budget constraints in crowdsourcing. In *HCOMP 2016*, 2016.

[53] D. Papailiopoulos, I. Mitliagkas, A. Dimakis, and C. Caramanis. Finding dense subgraphs via low-rank bilinear optimization. In *International Conference on Machine Learning*, pages 1890–1898. PMLR, 2014.

[54] J. Pons, J. Serrà, and X. Serra. Training neural audio classifiers with few data. In *ICASSP*, pages 16–20, 2019.

[55] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[56] T. K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13(1):23–52, 1988.

[57] H. Singh, M. Kumar, and P. Aggarwal. Approximation of heaviest k-subgraph problem by size reduction of input graph. In *ICCCN*, pages 599–605, 2019.

[58] M. S. Sorower. A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*, 18, 2010.

[59] R. Sotirov. On solving the densest k-subgraph problem on large graphs. *Optimization Methods and Software*, 35(6):1160–1178, 2020.

[60] C. Sun, N. Rampalli, F. Yang, and A. Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *PVLDB*, 7(13):1529–1540, 2014.

[61] J. Tang, Q. Chen, M. Wang, S. Yan, T.-S. Chua, and R. Jain. Towards optimizing human labeling for interactive image tagging. *TOMM*, 9(4):29, 2013.

[62] R. Taylor. Approximation of the quadratic knapsack problem. *Operations Research Letters*, 44(4):495–497, 2016.
[63] T. Tian, N. Chen, and J. Zhu. Learning attributes from the crowdsourced relative labels. In *AAAI*, volume 1, page 2, 2017.
[64] D. Vandic, F. Frasincar, and U. Kaymak. A framework for product description classification in e-commerce. *J. Web Eng.*, 17(1&2):1–27, 2018.
[65] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.
[66] Y. Wang, Y. E. Xu, X. Li, X. L. Dong, and J. Gao. Automatic validation of textual attribute values in e-commerce catalog by learning with limited labeled data. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2533–2541, 2020.
[67] G. M. Weiss and F. Provost. Learning when training data are costly: The effect of class distribution on tree induction. *J. Artif. Int. Res.*, 19(1):315–354, Oct. 2003.
[68] G. M. Weiss and Y. Tian. Maximizing classifier utility when there are data acquisition and modeling costs. *Data Min. Knowl. Discov.*, 17(2):253–282, 2008.
[69] M.-C. Yuen, I. King, and K.-S. Leung. A survey of crowdsourcing systems. In *SocialCom/PASSAT*, pages 766–773, 2011.
[70] T. Zahavy, A. Krishnan, A. Magnani, and S. Mannor. Is a picture worth a thousand words? A deep multi-modal architecture for product classification in e-commerce. In *AAAI*, pages 7873–7881, 2018.
[71] Y. Zhang and Q. Yang. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017.
[72] Y. Zhang and Q. Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2018.