Classifier Construction Under Budget Constraints

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.

CCS CONCEPTS

• **Information systems** → *Incomplete data*; *Clustering and classification*; • **Theory of computation** → *Approximation algorithms analysis.*

KEYWORDS

Classifier construction; Attributes extraction; Data completion;

ACM Reference Format:
Shay Gershtein, Tova Milo, Slava Novgorodov, and Kathy Razmadze. 2022. Classifier Construction Under Budget Constraints. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3514221.3517863

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517863

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 [19, 23, 24] 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. Continuing with the example of an e-commerce platform, consider in addition to the "wooden table" 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 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.

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 to have improved results pertaining to running shoes, than both table queries. The prioritization of queries is 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).

Applications. The motivation for our work is optimizing the effectiveness of classifier construction to recover missing data and metadata. Thus, our work can also be classified as optimizing (meta)data curation/cleaning or attribute extraction. A common use case for recovering missing metadata is improving search results, which for large companies may drastically improve profits. Hence, in our empirical analysis, utilities capture the importance of specific search queries. More generally, companies build dedicated high-accuracy classifiers to recover specific missing properties that are then stored in a database. This structured data is then leveraged not only to improve free-text query results but also other search and categorization tasks that are less noise-tolerant, such as faceted search [64] or providing the user with more complete information when viewing a description of an item. Our abstract model is agnostic to how utility is assessed and, therefore, the utility estimates may also take into account these broader objectives.

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 [19, 23, 24] 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 [14, 17, 30] (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 hardness bounds above, works well in practice, we leverage the high prevalence of short queries in real-life workloads demonstrated in [24] 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 *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 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, to find the classifier set of minimum cost that reaches it. We show that our analysis methods can also be applied to derive complexity bounds and algorithms for these two problems.

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 present definitions and results that will prove useful in our theoretical analysis.

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 the 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 $\mathcal{U} : Q \mapsto \mathbb{R}_+$. 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 $\langle Q, \mathcal{U}, C, B \rangle$.

[1]Compared to multi-valued classifiers, binary classifiers have higher accuracy, and are preferred when accuracy is essential [58].

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. Note that, in general, the budget may not be sufficient to cover all queries. The objective of *BCC* is to find a classifier set of maximum utility in this solution space. More formally, we aim to compute $argmax_{S \subseteq CL, C(S) \leq B} \sum_{q \in Q(S)} \mathcal{U}(q)$.

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, it is arguably not trivial to quantify these a priori. Hence, as in [24], the overall cost of a classifier set is the sum of the individual costs.

We also follow [24] 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 [32]. Moreover, estimating in advance the relative utility of such partial covers out of the overall utility of the query is challenging.

For a solution to be judged suitable by typical e-commerce companies classification accuracy must reach a high threshold for the full multi-faceted search criteria (in our experiments, discussed in Section 6, analysts employed by our industry collaborators provided cost estimates based on the experience of training classifiers to exceed 95% accuracy). Therefore queries that are not covered by a solution are considered not meeting the company's quality standard. The practical quality of a solution is thus respectively reflected by of weighted sum of the covered queries in our model.

We leave to future work the study of models that account for overlaps in construction costs, partial covers that provide quantifiable value, or multiple accuracy thresholds.

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

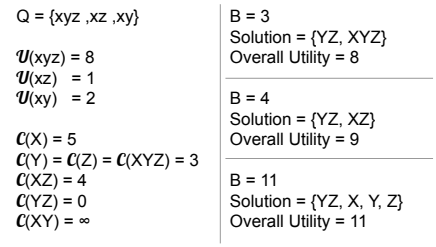

Figure 1: Three examples of *BCC* problem instances. The left side depicts the queries, utilities and costs, shared by all instances. The three different budget values, and the optimal solutions are depicted on the right side.

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 [28]). 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, since its cost exceeds the budget. 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.

Note that, in our model, we can always assume, for all classifiers considered in the solution space, that $C(XY) < C(X) + C(Y)$. Moreover, if $C(XY) \geq C(X) + C(Y)$, then XY may be safely pruned without affecting the optimality. This is because any solution that uses XY can instead use X and Y, which may only improve the coverage without increasing the cost. Therefore, the convention is to assign the cost of infinity for any pruned classifier. The infinite cost means that this classifier should not be considered as part of the solution space and does not mean that it is impossible to construct this classifier for a finite cost. Specifically, if the cost of XY is estimated at best to be $C(X) + C(Y)$, then we assign to it an infinite cost, which means that the algorithm will never examine or select it, as there is a provably better alternative.

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. Recall that the utility of the solution is the sum of the utilities of the covered queries. With xyz 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.

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

We next overview the problem studied in [24], 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 that covers all queries.

Theorem 2.5. *[24] 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 admits $\min\{2^{l-1}, O(\log n)\}$ approximation.*

A more central role in our analysis is played by graph and hypergraph density problems, as defined next.

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. *[7, 8, 62] All four problems 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 [20, 39], there remains a large gap between the proven hardness of *DkS* and the best known approximation factor, $\tilde{O}(n^{1/4})$ [8]. Under widely-believed complexity assumptions, it cannot be approximated to within a constant factor [4]. 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* [14, 17, 30], as the "Dense vs Random" conjecture [18, 48] implies that the $\tilde{O}(n^{1/4})$ factor is tight. The discussion above also roughly applies to *DkSH* [5].

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 (all formal proofs appear in our technical report [3]).

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

For $BCC_{l=2}$, we prove that it generalizes *DkS* (Definition 2.6), and is, thus, 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 are uniform, the inapproximability of the problem is retained.*

Lastly, we would like to note the robustness of the hardness result above, focusing for simplicity on the case of $l = 2$. The proof of Theorem 3.3 demonstrates that even if we extend I_2 such that the cost of every classifier XY satisfies $C(XY) \geq \gamma(C(X) + C(Y))$ for any $\gamma = o(poly(n))$, then the hardness bound is relaxed by at most a negligible γ factor. Therefore, while the hardest inputs pertain to solutions that consist mostly of singleton classifiers, this hardness bound remains more or less the same, even if we assume that most classifiers of length 2 cost polylogarithmically less than the corresponding singleton classifiers.

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 [24] 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 [24] (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 [24], 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 $l = 2$

Overview. In this subsection, we first explain how we partition $BCC_{l=2}$ into two problems: one is equivalent to the Knapsack problem, and the other - to the QK problem (Definition 2.6), which is much harder to approximate in the worst case. We first present a algorithm for QK focused on the worst case approximation, with no regard for scalability. We then modify it to use HKS solvers to derive practical running time and solution quality for typical inputs, without catering to artificial worst-case inputs. Then, in the next subsection, we will present an algorithm for $l > 2$, that repeatedly employs the algorithm described in this subsection, to solve the problem in iterations, in concert with two pruning heuristics to further improve scalability.

Approach 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 instance, with the optimal solution to one of these problems yielding at least half of the optimal utility of the original BCC instance (we will show that the union of the solution spaces of these two sub-instances is exactly the original solution space, hence, at least half of the optimal solution, in terms of utility, is a valid solution of one of the two sub-instances). 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 $\{X, Y, Z\}$).

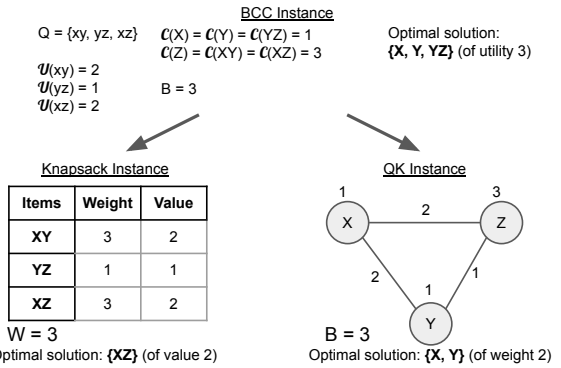

Figure 2: Example of a BCC instance with $l = 2$, separated into Knapsack and QK instances. The optimal solution of each instance is depicted next to the corresponding input.

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.

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 queries, 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 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 $BCC_{l=2}$.*

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.

To provide context for the modifications in A_H^{QK}, we first briefly outline the high-level approach of A_T^{QK}. Concretely, A_T^{QK} partitions the graph into $O(\log^3 n)$ subgraphs (to solve each separately) of a simple form: a bipartite graph (left and right node sets), with uniform edge weights, where all costs on the left side are 1, and all costs on the right side are $w > 1$. Then, each node on the right side is replaced by w copies of weight 1, and a $\tilde{O}(n^{1/4})$ *DkS* algorithm is employed. Finally, a greedy procedure transforms the *DkS* output into a solution over the original graph.

We will use in A_H^{QK} the idea of replacing each node with multiple copies, however, we do so in a more general graph setting, and thus use a more involved novel procedure to transform the *DkS* solution over this graph into a QK solution over the original graph.

Effect of cost distributions on hardness. Before presenting our heuristic QK algorithm, we first note why, in our examination of easier special cases, we only target the case that can be efficiently solved via a Knapsack algorithm, and are not examining different cost structures pertaining to the QK instance.

First, note that, since we can effectively solve the Knapsack subproblem, the worst-case hardness pertains only to instances where no good approximate solution contains a non-negligible subset that corresponds to the Knapsack solution. That is the hard cases must require solutions that are almost entirely derived via the QK ($BCC(2)$) subproblem, otherwise our algorithm guarantees a good approximation. Therefore, when aiming to bypass the worst-case hardness we may focus only on the subset of the input consisting of queries of length two and singleton classifiers. Theorem 3.3 shows that QK is hard even when all costs are uniform. One may wonder, however, whether some reasonable assumptions on the cost distribution may relax the worst-case bound. Note that the worst-case hardness of $\tilde{O}(n^{1/4})$ still applies to any cost structure that can be derived from the uniform distribution by multiplying or dividing the costs by factors that are upper-bounded by $\gamma = o(poly(n))$, which is the case for a small constant γ in all our examined datasets. This is a simple corollary that follows from the result we prove in Section 4 of our technical report [3]: changing the budget by a γ factor changes the optimal score by at most a γ^2 factor. Nevertheless, we bypass the worst-case hardness by using a modern *HkS* solver [41] that is shown to achieve close to optimal performance over inputs with uniform singleton costs, and we prove below in Theorem 4.7 that our adaptation to varying weights adds at most a small constant factor.

Heuristic QK algorithm. The first modification we perform in A_T^{QK} is replacing the worst-case-oriented *DkS* algorithm 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.

Due to space constraints, we show here a simplified version of A_H^{QK}, omitting the more technical phases. The full algorithm appears in our technical report [3], where we also prove the following worst-case bound on its performance, which is conditioned on the performance of the *HkS* algorithm (that was shown in [41] to provide solutions close to 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 show in our proofs 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.

Preprocessing. We also omit here our preprocessing procedure that transforms the $BCC_{l=2}(2)$ input, such that all classifier costs (and thus all node costs in the QK instance) are integers in $[1, B/2]$. We assume henceforth that the input is already transformed as mentioned above.

The simplified A_H^{QK} algorithm is depicted in Algorithm 1.

Random bipartite graph (lines 1-3 in Algorithm 1). Given as input a budget B and a graph $G = (V, E)$, with node costs given by $C(\cdot)$ and edge weights given by $\mathcal{W}(\cdot, \cdot)$, the first step is to transform G into a bipartite graph. To do so, we adopt a randomized procedure pointed out in [53]. we employ it $\log n$ times (we do so in parallel) and choose the best solution of all iterations (line 1). In each iteration, we partition V into two sets $\bar{L}$ and $\bar{R}$, assigning each node independently to one of the sets with uniform probability (line 2). 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 (line 3). With

Algorithm 1: A_H^{QK}

Input: budget B; graph $G = V, E$, with costs $C : V \mapsto [B/2]$ and weights $\mathcal{W} : E \mapsto \mathbb{R}_+$

1 repeat $\log n$ times the following algorithm, and choose the best solution:
2 assign each node in V independently with uniform probability to one of the two sets $\{\bar{L}, \bar{R}\}$
3 derive from G a new graph $\bar{G} = (V, \bar{E})$ where $\bar{E} = E \cap (\bar{L} \times \bar{R})$, with the same costs and weights
4 derive $\hat{G} = (\hat{V}, \hat{E})$ from $\bar{G}$, as follows:
5 replace every $v \in V$ with $C(v)$ copies
6 $\forall \{v, u\} \in \bar{E}$: connect all copies of v and u with edges of weight $\frac{\mathcal{W}(v,u)}{C(v) \cdot C(u)}$
7 denote the set of copies of $\bar{L}$ and $\bar{R}$ by $\hat{L}$ and $\hat{R}$, respectively
8 run an HkS algorithm over $\hat{G}$ with $k = B/2$, denoting the output by $\hat{S}$
9 let $L = \hat{L} \cap \hat{S}$ and $R = \hat{R} \cap \hat{S}$
10 denote by $V_L \subset V$ the nodes of G whose copies are in L
11 $\forall v \in V_L$: let $cop_L(v)$ denote the number of copies of v in V_L
12 solve the following Knapsack instance, denoting the output by $\tilde{R}$:
13 the items are V/V_L
14 the weight of each v is $C(v)$
15 the value of each v is $\sum_{u \in V_L} (\mathcal{W}(v, u) \cdot \frac{cop_L(u)}{C(u)})$
16 the weight bound is $B - \sum_{u \in V_L} cop_L(u)$
17 solve the following Knapsack instance, denoting the output by $\tilde{L}$:
18 the items are $V/\tilde{R}$
19 the weight of each v is its cost
20 the value of each v is $\sum_{u \in \tilde{R}} \mathcal{W}(v, u)$
21 the weight bound is $B - \sum_{v \in \tilde{R}} C(v)$
22 return $\tilde{L} \cup \tilde{R}$

probability exceeding $(1 - 1/n)$ (following the proof in [53]), 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.

Solving HkS on a blown-up graph (lines 4-9). 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 $\bar{G}$ (line 4), as follows. Each node $v \in V$ is replaced in $\hat{V}$ by $C(v)$ copies (line 5), 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 $\frac{\mathcal{W}(v,u)}{C(v) \cdot C(u)}$ (line 6). The sets of copies of $\bar{L}$ and $\bar{R}$ are denoted in $\hat{G}$ by $\hat{L}$ and $\hat{R}$, respectively (line 7). We then run an HkS algorithm over $\hat{G}$ with $k = B/2$, yielding a solution $\hat{S}$ (line 8) (we allocated only half of the budget, thus losing a factor of 2 in the optimality, as the other half is used when transforming $\hat{S}$ into a solution over the G.) Given $\hat{S}$, we further use $L = \hat{L} \cap \hat{S}$ and $R = \hat{R} \cap \hat{S}$ (line 9), 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 $\mathcal{W}(v, u)$. 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.

Knapsack instances (lines 10-22). We next use two Knapsack procedures to derive a subgraph of G, of cost at most B, whose weight is at least a 4/5-fraction of the weight of the subgraph induced by $\hat{S}$ in $\hat{G}$, up to an ϵ factor in the Knapsack approximation. For this, we select arbitrarily the set L, and denote by $V_L \subset V$ the nodes of G whose copies are in L (line 10). For each $v \in V_L$, we denote the number of copies of v in L by $cop_L(v)$ (line 11). We then solve the following Knapsack instance (line 12): the items are the nodes V/V_L (line 13); their weights are their costs (line 14); the value of each node v is $\sum_{u \in V_L} (\mathcal{W}(v, u) \cdot \frac{cop_L(u)}{C(u)})$ (line 15), which is the sum of the edge weights in $\hat{G}$ that connect all copies of v to L; and the weight bound is the budget after deducting the selection of L: $B - \sum_{u \in V_L} cop_L(u)$ (line 16). The output is denoted by $\tilde{R}$, which, at this point, replaces R. To also replace L with nodes from G, we solve another Knapsack instance (line 17): the items are $V/\tilde{R}$ (line 18); the weight of each node is its cost (line 19); the value of each node v is $\sum_{u \in \tilde{R}} \mathcal{W}(v, u)$ (line 20), the sum of edge weights connecting it to R; and the weight bound is what is left of the budget after selecting $\tilde{R}$: $B - \sum_{v \in \tilde{R}} C(v)$ (line 21). Denoting the output of the Knapsack algorithm by $\tilde{L}$, the final output is $\tilde{L} \cup \tilde{R}$ (line 22).

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.

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 $MC3$ problem (Definition 2.4), for which we can use the algorithm from [24] (Theorem 2.5). The fraction of the budget saved by the solution to the $MC3$ 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 next.

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.

Budget allocation. We, therefore, initially use a constant fraction 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*.

Determining the exact fraction of the budget allocated to the first iteration is somewhat arbitrary. It is important to note, that the subsequent iterations also solve only the $BCC(1)$ and $BCC(2)$ subproblems. No stage of the algorithm solves any $BCC(i)$ instance for $i > 2$. Additionally, queries of length 1 or 2 are targeted by the algorithm in all iterations. In particular, typical *BCC* inputs primarily consist of queries of length at most 2, hence, the algorithm "focuses" almost exclusively on these short queries in **all** iterations, regardless of the initial budget allocation. The budget allocation only determines how quickly some of the rare longer queries are integrated into the examined solution space. For this reason, the exact allocation is to a large extent inconsequential over practical inputs. Our experiments verify the small effect of the allocation choice (especially, for any fraction in $[0.2, 0.9]$) on the overall score, even on a synthetic dataset with a higher percentage of longer queries. The results and explanation of this invariability phenomenon, are provided in Section 6. We, therefore, allocate to the first iteration of the algorithm half of the budget, as this is one of the best-performing values over all datasets. Nevertheless, this fraction may be considered as a parameter, and over other datasets, it may be beneficial to test different allocations.

Pruning optimization. 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.

Algorithm. We next list the steps of the algorithm outlined above, as depicted schematically at high-level in Algorithm 2.

Algorithm 2: A^{BCC} (high-level scheme)

1 preprocessing: apply two pruning methods to reduce the number of classifiers
2 allocate half of the budget to solve the $BCC(1)$ 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 [24]
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 2). 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, in instances with uniform costs, the solution space is reduced to using only singletons. Note, however, that in some edge cases, this pruning rule is not applied. Concretely, for any given query, if the rule would retain only short classifiers, such that any combination of these classifiers that covers the query exceeds the budget, then we do not prune the longer classifiers relevant to this query. The second pruning procedure is based on *weighted leverage scores* [11, 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 [24], 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 this section, we define two alternative objectives for which our *DKS*-based analysis methods yield hardness bounds and algorithms. These problems, defined below, may be of interest in practical scenarios where there is some flexibility in the budget constraint,

Definition 5.1. In the *Generalized MC*3 *problem* (*GMC*3) the input is $\langle Q, \mathcal{U}, C, T \rangle$, where Q, $\mathcal{U}$ 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 $\langle Q, \mathcal{U}, C \rangle$, where Q, $\mathcal{U}$ 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.

Due to space constraints, the detailed descriptions of our theoretical and empirical results for these problems are deferred to our technical report [3]. Concretely, we provide hardness bounds and an algorithm for *GMC*3, showing that despite it containing *MC*3 (Definition 2.4) as a special case, the problem is more similar theoretically to *BCC*. In particular, we show that at a worst-case cost of a $\log n$ factor in the performance ratio, our *BCC* algorithm can be adapted to *GMC*3. 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.

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. We start with describing our experimental setup and then present the evaluation results.

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 [24]. 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, 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 [19, 24] 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. 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 ∞ 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 and the search frequency of 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 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 [28]. 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 terms of adapting *MC*3 algorithm to *BCC*, note that of all works on *MC*3, only [24] provides a relevant *MC*3 algorithm, as [23] uses the same algorithm, which subsumes the algorithm in [19]. Observe that we cannot use any *MC*3 algorithm directly as ignores the budget constraint. We only use the *MC*3 algorithm to determine the upper bound on the range of examined budgets. Nevertheless, 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 [24]. This is essentially the same greedy algorithm, except it terminates as soon as the budget bound is reached.

- **RAND** - This simple baseline randomly selects in each iteration one of the classifiers whose selection will not exceed the budget.

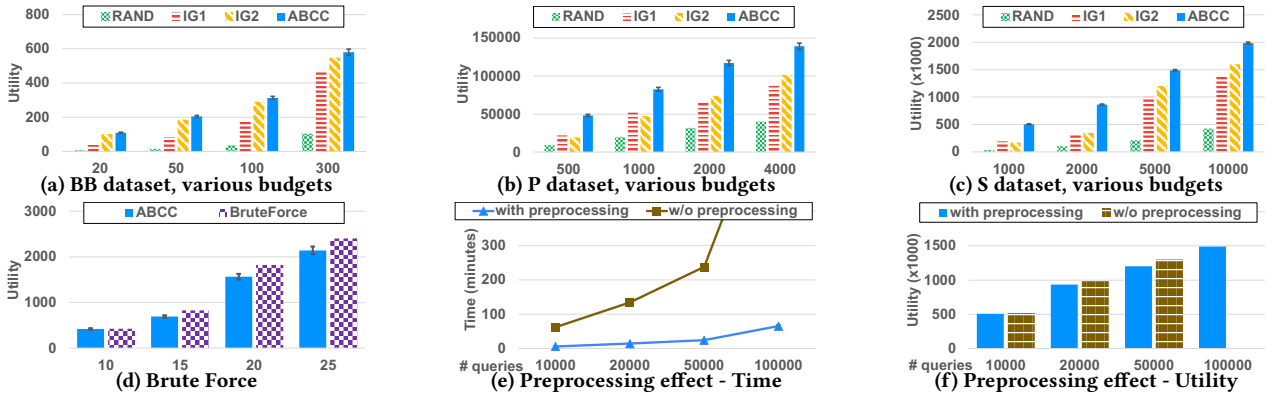

Figure 3: Experimental results

- **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 2 in Section 4).

Evaluation outline. We compared the algorithms listed above over all datasets, both in terms of the overall utility of the selected classifier set and the overall running time. We executed the randomized algorithms, which are A^{BCC} (due to step 2 in Algorithm 1) and the *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 [24], 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. We also examined how different budget allocations in step 2 of Algorithm 2 affect the score. Lastly, we examined the effect of our preprocessing step (the first step of Algorithm 2) on the execution time and the solution quality.

6.2 Evaluation Results

We next present the results 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. Moreover, the variance of our randomized A^{BCC} algorithm across 5 executions was very low (since each execution of Algorithm 1, employed by A^{BCC}, already takes the best solution out of multiple iterations): less than 3% over the *BB* and *P* datasets, and less than 1% over the *S* dataset. This is visualized in the figures via error bars, which are either barely visible, or not visible at all (over the *S* dataset).

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 over a subset of the *P* dataset (we tested small query subsets that pertain to very specific subdomains). 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 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 *DKS* case, where the number of edges is quadratic in the subgraph size), but rather sublinear. This is because most of the utility is typically concentrated in much smaller subsets of the instance.

One important corollary is that compared to the budget required to cover all queries (which is computed in the *MC*3 setting of [24]) 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 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 is roughly 2000, and this is sufficient for 65% of the total utility. This loose bound on the optimal utility implies that our performance ratio is at most 4/3 for 50% of the budget. We also note that the total utility possible over the *BB* dataset is roughly $1K$, whereas over the *S* dataset it is roughly $2.5M$.

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, 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 2).

Budget Allocation. Recall that A^{BCC} allocates half of the budget to the first iteration. As discussed in Section 5, this allocation is a heuristic. Examining a wide range of fractions in $[0, 1]$ over all datasets, indicated that this choice affects the score by less than 5%. The only significant trend was that the best choice lies in the range $[0.2, 0.9]$, and values in this range typically differ in the score by less than 1%. As discussed in Section 4, the small effect on the solution is due to the fact that the queries of length at most 2 are targeted by all iterations of the algorithm. In general, once a query is integrated into the solution space, every subsequent iteration will also target this query unless it is already covered.

The reason the algorithm performs worse when the allocation fraction is close to 0 is that the residual problem in the second iteration is too similar to the original problem, and longer queries are not considered. Similarly, the performance typically worsens for fractions close to 1 since the budget remaining for the subsequent iterations is rather small. Nevertheless, even when all the budget is used on the first iteration, the performance degradation is limited, since most of the queries are of length at most 2.

Scalability and Preprocessing. Finally, we discuss scalability, and the effect of our preprocessing procedure (step 1 in Algorithm 2), which reduces the size of the input. 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 budgets, 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 100K 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. 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} 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. We also report that all other examined baselines are much faster. However, since this is an offline task that aims to derive the most cost-effective plan for corporations to increase profits, a slower algorithm that increases the solution quality is arguably preferable. We note that the performance also benefits from the total parallelizability of step 1 in Algorithm 1.

Lastly, recall that the preprocessing procedure described in Section 4 consists of two independent pruning heuristics. The experiments have shown that the effect of both heuristics on the quality is equally negligible, whereas, in terms of the speed-up, roughly 80% is due to the heuristic based on the leverage scores.

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 [19, 23, 24]. 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 $MC3$ [24] 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 [27, 36], identifying helpful sentences from user reviews [22, 35], fraud detection [6, 37, 49], finding a proper category for an item in the company's taxonomy [33, 65, 72], and classifying search queries [12], which is also the focus of our work. As mentioned in the introduction, classifier construction is known to be expensive [68, 69], 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 [21, 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 [15]. 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, 67] 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 [29]). 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 [71], 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) [31], and that the execution time of each query is counted towards the objective function regardless of the view selection [25]. 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* [25, 26] 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 [9, 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, 73, 74]. 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* [13, 34], 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 [24] 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 [9], and one can also consider superpolynomial algorithms [10].

8 CONCLUSION AND FUTURE WORK

In this work, we studied 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 multiple accuracy thresholds and utility in partial covers of queries, or generalizing the cost function to capture overlaps in classifier construction. Moreover, our methods may be adapted to other similar problems where one must select a set of components such that their subsets can be combined to address a set of objectives (e.g., the *MVS* and *MTL* problems, discussed in Section 7). Lastly, our scalable heuristic *QK* algorithm may be applied to problems that also reduce to *QK*, such as many entity summarization works [16, 44, 70] or incremental view maintenance [75] where a *QK* formulation is avoided due to inefficiency.

Acknowledgment. We are grateful to the anonymous reviewers, for their insightful comments. This work has been partially funded by the Israel Science Foundation, BSF - the Binational US-Israel Science foundation and the Tel Aviv University Data Science center.

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] Technical Report. http://slavanov.com/research/bcc-tr.pdf.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

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

[16] G. Cheng, D. Xu, and Y. Qu. C3d+ p: A summarization method for interactive entity resolution. *Journal of Web Semantics*, 35:203–213, 2015.

[17] 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.

[18] 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.

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

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

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

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

[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] Q. Liu, G. Cheng, K. Gunaratna, and Y. Qu. Entity summarization: State of the art and future challenges. *Journal of Web Semantics*, page 100647, 2021.

[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. Springer, 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. Tunkelang. Faceted search. *ICR*, 1(1):1–80, 2009.

[65] 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.

[66] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[67] 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.

[68] 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.

[69] 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.

[70] D. Xu, L. Zheng, and Y. Qu. Cd at ensec 2016: Generating characteristic and diverse entity summaries. In *SumPre@ ESWC*, 2016.

[71] M.-C. Yuen, I. King, and K.-S. Leung. A survey of crowdsourcing systems. In *SocialCom/PASSAT*, pages 766–773, 2011.

[72] 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.

[73] Y. Zhang and Q. Yang. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017.

[74] Y. Zhang and Q. Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2018.

[75] W. Zhao, F. Rusu, B. Dong, K. Wu, and P. Nugent. Incremental view maintenance over array data. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 139–154, 2017.