Minimization of Classifier Construction Cost for Search Queries

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

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

Gefen Morami
Tel Aviv University
gefenkeinan@mail.tau.ac.il

Slava Novgorodov
eBay Research
snovgorodov@ebay.com

ABSTRACT

Search over massive sets of items is the cornerstone of many modern applications. Users express a set of properties and expect the system to retrieve qualifying items. A common difficulty, however, is that the information on whether an item satisfies the search criteria is not explicitly recorded in the repository. Instead, it may be general knowledge or "hidden" in a picture/description, leading to incomplete search results. To overcome this problem, companies build dedicated *classifiers* that determine which items satisfy the given criteria. However, building classifiers requires volumes of high-quality labeled training data. Since the costs of training classifiers for different subsets of properties can vastly differ, the choice of which classifiers to train has great monetary significance. The goal of our research is to devise effective algorithms to choose which classifiers one should train to address a given query load while minimizing the cost.

Previous work considered a simplified model with uniform classifier costs, and queries with two properties. We remove these restrictions in our model. We prove NP-hard inapproximability bounds and devise several algorithms with approximation guarantees. Moreover, we identify a common special case for which we provide an exact algorithm. Our experiments, performed over real-life datasets, demonstrate the effectiveness and efficiency of our algorithms.

ACM Reference Format:
Shay Gershtein, Tova Milo, Gefen Morami, and Slava Novgorodov. 2020. Minimization of Classifier Construction Cost for Search Queries . In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3318464.3389755

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389755

1 INTRODUCTION

Search over massive assortments of items is the cornerstone of many modern applications, including, e.g., online stores, video streaming services, news article archives, etc. Users often express (through a variety of user interfaces, ranging from form-based to free-text queries) a set of properties that the desired items should satisfy, and expect the system to retrieve qualifying items. A common difficulty however is that the information on whether or not an item satisfies the search criteria is sometimes not explicitly recorded in the repository. Instead, it may be "hidden", e.g., in the picture/textual description associated with the item, or as general knowledge not recorded in the database, thereby leading to incomplete search results. For example, product catalogs in popular e-commerce sites often contain incomplete information as sellers tend to focus, in their provided information, only on the most important properties, such as product name and cost, with additional information given through natural language description and/or a picture. The fact that a given clothing item is, e.g., a "white summer dress", may be derivable only from its picture (for the color and sleeveless shape) and its textual description (for the cloth material).

To overcome this problem, there is an ever growing trend of companies leveraging Machine Learning (ML) algorithms to complete the missing information. In particular, much effort is put into building dedicated *classifiers* that allow to determine whether an item satisfies the given search criteria or not [46]. However, building such accurate classifiers typically incurs non-trivial cost due to the volumes of high-quality labeled training data that is needed: labeling is often performed by humans for each data item separately, leading to bottleneck concerns, particularly when expertise is required, as there are fewer experts and their time is more valuable. Thus optimizing the classifiers construction process is an important goal.

We focus in this work on a common class of conjunctive search queries that test the satisfaction of a set of properties. Note that given a set Q of such queries, there are multiple sets of classifiers that may be built to answer Q. At the one extreme, one can build a dedicated classifier for each query in Q. At the other extreme, one can build a classifier for each of the individual properties tested in Q, then conjunct their output to evaluate the queries. Midway solutions are also

Shirts

pr_id	pr_title	description	price	team	color	brand
Ju18W1	Juventus White 18/19	Juventus shirt from 2018/19 season, ...	$74.99			Adidas
Ch17B2	Chelsea Blue #18	Chelsea Olivier Giroud #18 shirt, ...	$64.90	Chelsea		
Ar19W1	Arsenal Red Long Sleeve	Arsenal 2017/18 red shirt with long sle...	$84.90		Red	
CS19Re	CSKA Moscow #14	CSKA Moscow shirt, Nababkin #14	$69.90	CSKA		Umbro

Figure 1: 'Shirts' relation example.

possible, as exemplified below, where classifiers are built for (the conjunction of) selected subsets of the properties, then assembled to answer Q. The choice of which classifiers set to build depends on the training cost of the different classifiers, pertaining to different subsets of properties, which can vastly vary. The goal of our research is to devise effective algorithms to choose which classifiers one should train to be able to address a given query load, while minimizing the overall construction cost.

We consider here binary classifiers. These are often preferred in practice over multi-valued classifiers, as the increased granularity is essential for accurate performance. Furthermore, typical methods for building *multi-label classifiers* often consist of independently training a binary classifier for each label [45]. Nevertheless, our model can capture multi-valued classifiers as well. We explain in Section 5 how the input can be adapted to account for these extra options, so that our approach applies all the same.

Before presenting an overview of our approach and results, let us first illustrate through a simple example how carefully selecting which classifiers to construct can lead to more cost-efficient solutions.

Example 1.1. Consider an e-commerce website where sellers upload soccer shirts. The website has a products database (depicted in Figure 1) which includes general information such as product title, price and image. In addition, each shirt has a set of attributes (e.g., brand), either provided by the seller, or derived automatically from the title, description and image (these attributes have grey background in the figure). Consider the following two free-text search queries: "white adidas juventus shirt" and "adidas chelsea shirt". The queries are translated by the e-commerce application (e.g., via NLP-based methods) into the following SQL expressions, to be evaluated on the database:

```
SELECT * FROM Shirts WHERE `team` = 'Juventus'
AND `color` = 'White' AND `brand` = 'Adidas';

SELECT * FROM Shirts WHERE `team` = 'Chelsea'
AND `brand` = Adidas';
```

Accurately answering the queries depends on correctly classifying each item w.r.t. whether or not they satisfy the color, brand and team requirements. To answer the first query one may train three binary classifiers (which, for brevity, we denote by the initials of the properties): a J classifier that tests whether the shirt's team is Juventus, a W classifier that tests whether the shirt's color is white, and an A classifier that tests whether the brand is Adidas. One can also train classifiers for any combinations of these properties, e.g., a AJ classifier (tests whether an item is a Juventus shirt of Adidas brand), a AW classifier, or alternatively a full JAW classifier which test the entire query. Similarly, for the second query one could train the classifiers: C (Chelsea), A or CA. The choice of which classifiers to build depends on their construction cost.

The choice of which classifiers are more cost-effective to build depends on the individual classifier costs and which queries they can serve. Note that while both queries need to know whether or not the shirt's brand is "Adidas", building this single-property classifier may not be the best choice. Indeed, while it may seem counter-intuitive, in some cases the cost of training a multi-property classifier (e.g., "X and Y classifier") may be cheaper than the sum of the corresponding single-property ones (e.g. "X classifier" and "Y classifier"). For instance, detecting that a shirt is an Adidas shirt may be non-trivial and require many labeled samples (due to the variety of the Adidas shirt types that are used by different teams). Similarly, general detection of Juventus shirts may be challenging (as the team switched multiple designs and sponsors in the last decade). In this example classification for the "Adidas Juventus" conjunction is an easier task, since these shirts have just a few variants.

Assume that the classifier construction costs (for a given accuracy level) are as follows, for some cost unit N:

```
C: 5N,  A: 5N,  J: 5N,  W: 1N,
AC: 3N, AW: 5N, AJ: 3N, JW: 4N, JAW: 5N
```

The optimal set of classifiers to build, given these queries and costs, is $\{AC, AJ, W\}$, whose cost is $7N$. We now briefly discuss the intuition behind this solution, which will later guide our algorithms. In general, the fewer properties the classifier tests the more queries it may potentially be used for. For example, the classifier A can be used to answer both queries, whereas the classifier AC can only be used to answer the second query. On the other hand, the latter costs less than the former, demonstrating a trade-off between cost and range of applicability. In this specific case, because A is only relevant for the first query, it is not more useful than the cheaper AC classifier. As the same holds for AJ, choosing AC and AJ turns out to be preferable to choosing A, J and C individually. Finally, the choice of W is not surprising since its cost is extremely low, and it is required to complete the answer to the first query.

The problem of selecting the minimal-cost set of classifiers to cover a given query load has been initially introduced by [13] in a short vision paper, but the proposed model and algorithms there were based on two over-simplifying assumptions. First, it was assumed that construction costs are identical for all classifiers. Second, the number of properties tested by a query, which we refer to henceforth as the *length* of the query, was assumed to be at most two. Accordingly, the limited setting has lead to straightforward algorithms. In practice, real-life scenarios are more complex. First, the monetary cost of training a given classifier can be estimated in advance [44], and analysis of real-world applications and data shows that these costs can vary immensely (see Section 6). Second, while in practice the incidence of queries of a given length tends to correlate negatively with said length, it is nevertheless the case that queries whose length exceeds two make up a significant percentage of the overall query load. In this paper we address the general case of non-uniform classifiers costs and queries of arbitrary length. We prove this generalized model to principally differ from its restricted predecessor in [13] in terms of computational complexity, and, respectively, our optimization algorithms bear no resemblance to those in [13].

More formally, in the problem we study here, which we term as the *Minimization of Classifier Construction Cost for Search Queries* problem (MC^3), we are given a set of queries, each testing the satisfaction of a set of properties, and a list which associates a cost[1] with every binary classifier pertaining to a conjunction of any combination of one or more properties. Each such classifier can determine for its associated properties whether they all hold true (simultaneously) or not. We are interested in training a set of classifiers which can cover all queries. That is, for every query, the truth value of its conjunction of properties can be determined (w.r.t. any given item in the search space) via a conjunction of one or more classifiers out of those trained. The objective of the problem is to identify of all such possible sets of classifiers, one whose overall cost (the sum of the individual classifiers cost) is minimal.

We prove, via a reduction from the Set Cover problem, the NP-hardness of our problem, along with an inapproximability bound which grows stricter as the maximal query length increases. These hardness bounds hold even in a considerably restricted setting. Nevertheless, we demonstrate both in our theoretical and experimental analysis, that one can make headway, and provide effective algorithms for the problem. First, we study the special case where the maximal query length is two (but, unlike [13], the classifiers cost may still vary). We prove it to be a simpler problem, for which we provide an exact, PTIME solution. Concretely, we reduce this simple variant to a Max-Flow problem over bipartite graphs, thereby leveraging the best performing algorithms known for this special case of the problem. While, as explained before, in practice one often also encounters longer queries, there still exist some domains (exemplified in Section 6) where short queries are dominant, hence this special case has practical utility. Second, for the general problem, by reducing it to a special case of the Weighted Set Cover Problem (a different reduction from the hardness result), we identify two possible effective algorithms, both with approximation guarantees, each suiting a different range of query lengths. We further augment both algorithms with heuristics which preserve the approximation guarantees, yet improve in practice, as we demonstrate in our experiments, both the quality of the solution and the running time.

In our experiments (Section 6) we consider both synthetic data and real-world datasets provided by a large e-commerce company. These datasets include a curated set of queries (based on actual user sessions) along with actual cost estimations for training classifiers. We note that e-commerce is a particularly apt domain for the MC^3 problem, as the most popular sites have product catalogs of colossal proportions, often with incomplete information, as mentioned above. Consequently, these companies invest significant resources in developing classifiers which assist in query answering [46]. With e-commerce being a trillion dollar industry, even small improvements, enabling to present to users search results which they are more likely to be interested in purchasing, can greatly increase profits.

Our main contributions can be summarized as follows.

(1) We formulate the Minimization of Classifier Construction Cost for Search Queries problem (MC^3), accounting both for varying classifiers costs and query lengths.
(2) We study the computational complexity of MC^3 and prove its NP-hardness and approximation hardness.
(3) We identify the special case of queries of length at most two as solvable in polynomial time, and propose an exact algorithm for it, which can handle non-uniform classifiers costs. We show in our data analysis that in some domains such queries form over 95% of the query load, thus this special case is also relevant in practice.
(4) We propose efficient approximation algorithms to solve the general problem, based on a reduction to the Set-Cover problem, coupled with novel preprocessing heuristics. Moreover, all our proposed algorithms come with approximation guarantees.
(5) We present an extensive experimental study based on real-world datasets obtained from a large e-commerce site, as well as on synthetic data, demonstrating the effectiveness and efficiency of our algorithms.

[1]The cost may reflect monetary cost, or be based on other criteria, such as the number of data items required for training, the time that it is estimated to take an expert to label all data items, etc.

Paper outline. Section 2 provides the necessary definitions and formalism for our problem, along with useful theoretical results. Section 3 describes our preprocessing heuristics. Section 4 studies the MC^3 problem restricted to queries of length at most two. The theoretical study is then extended to the general case in Section 5. Experimental studies are presented in Section 6. Finally, the related work appears in Section 7. We discuss future work and conclude in Section 8.

2 PRELIMINARIES

We start by formally defining the problem that we study in this paper, and discussing important aspects of our model. We conclude this section by presenting results for related problems that will be useful in developing our algorithms and proving our hardness results.

2.1 Formal problem definition

Recall that in the motivating setting, as described in the introduction, we have a set of distinct queries, which we denote by Q, each specifying a conjunction of properties. Let P denote a universe of properties. A query $q \subseteq P$ is defined as a set of properties. For any query q let $C_q = 2^q \setminus \emptyset$ 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. Given any item in the search space, a binary classifier, which pertains to some specific set of properties, returns true if the item satisfies all these properties, and false otherwise. Given a set of queries Q, let $C_Q = \cup_{q \in Q} C_q$ denote the union of the power sets of all its queries (except for the empty set). This represents the universe of classifiers. For any subset $S \subseteq C_Q$, we define $P(S) = \cup_{C \in S_q} \cup_{p \in C} p$ as the set of all distinct properties appearing in members (classifiers) of S.

The input $\langle Q, W \rangle$ to the *Minimization of Classifier Construction Cost for Search Queries* problem (MC^3) consists of a set Q of n queries over P, and a weighting function $W : C_Q \rightarrow [0, \infty)$ (later in this section, when we discuss input representation, we note that to capture practical settings where classifiers of infinite weight are simply omitted from the input, we do not count these classifiers towards the input size). This function assigns a cost to each classifier. The output is a set $S \subseteq C_Q$ of classifiers. The weight of any set $S \subseteq C_Q$ is defined as $W(S) = \sum_{c \in S} W(c)$, which is the sum of the costs of all the classifiers. We say that a query q is *covered* by $S \subseteq C_Q$ 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. A set of queries Q is said to be covered by $S \subseteq C_Q$ if all its queries are covered by S. The goal of MC^3 is to output a set of classifiers of minimal weight s.t. it covers Q.

Additional notation. To simplify notation, we use x,y and z to denote properties, thereby denoting a query $\{x, y, z\}$ as xyz, whereas a classifier $\{x, y, z\}$, that tests for the conjunctive satisfaction of the properties (x, y, z), is denoted by XYZ. Additionally, for any given query set Q, let $k = k_Q$ denote the maximal length of a query in it. This is an important parameter of the problem for two reasons. First, we study separately the variant of the problem where $k = 2$. Second, for the general problem, our approximation lower and upper bounds are functions of k. In our analysis, unless stated otherwise, k is assumed to be a constant, and in practice it rarely even exceeds 5. Nevertheless, we provide results for super-constant k as well. 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 representation and size. Observe that C_Q does not include all possible classifiers corresponding to all subsets of P. For instance, if $P = \{x, y, z, u\}$ and $Q = \{xy, zu\}$, then $C_Q = \{X, Y, Z, U, XY, ZU\}$. Classifiers such as XZ are not included in C_Q, since they are not relevant to the solution of the problem. Concretely, since no query includes both x and z, the classifier XZ cannot be used to cover any query. Lastly, we assume, for simplicity, that P only includes properties which appear in at least one query in Q.

We next clarify some ambiguity regarding the size of the input. The weighting function W, in particular, can be represented in various ways (for example, one can save space using an enumeration scheme for the domain C_Q), each implying a somewhat different input size. However, for $k = O(1)$ the input size of both parts of the input (and thus overall) is in all cases $\Theta(n)$. Hence, for uniformity, we assume a simple representation, where the input size of W is treated as the sum of lengths of the classifiers in its domain (we ignore, for simplicity, factors such as the logarithmic number of bits needed to represent weights and properties). The upper bound on the overall input size pertains to the case where all queries are disjoint and of maximal length. The representation size of Q here is nk. For each query, there are $\binom{k}{i}$ classifiers of length i, and hence we have a known result for a telescopic sum, which implies that the sum of lengths of classifiers for a query is $k \cdot 2^{k-1}$. Hence, the overall representation size is at most $(nk \cdot (1 + 2^{k-1}))$, which is $\Theta(n)$ for $k = O(1)$. In practice, it is not always true that all theoretically relevant classifiers are considered. When the number of combinations of properties is too large it is common to only consider classifiers of length at most $k' < k$. Furthermore, some classifiers are not considered because it is hard to bound their cost in advance, or they are deemed infeasible (e.g., not enough training data available). In some cases one can efficiently

determine in a preprocessing step (see Sections 4 and 5) that some classifiers are always inferior to others and should not be considered. Our model accounts for these classifiers by setting their weight to ∞ (we assume that Q can be covered by a solution of finite weight, and disregard the trivial cases where this does not hold). However, as in practice such classifiers are simply omitted from the input, in our model as well we do not count these towards the input size. As mentioned, for constant k this makes no difference asymptotically on the size of the input.

Model characteristics. In our model we assume that the classifiers are constructed in parallel and their construction costs are independent. While in practice there may be some overlap, e.g., in terms of data labeling or crowd-worker time, it is arguably not trivial to quantify such overlap in advance. Hence, in our model the cost of each classifier is independent, and the total cost of a set of classifiers is derived as the sum of all the individual costs. The procedure of constructing classifiers, followed by completing the corresponding missing values in the database is performed offline [2]. Hence, which specific classifiers were constructed has no bearing on the running time of search queries over the completed database.

We note that in practice queries also often contain properties for which there already exists knowledge sufficient to accurately determine in real-time whether they hold for any given item. To ensure consistency with our model, we assign a cost of zero for any classifier testing a property (or a conjunction of such properties) for which a classifier construction is not necessary. Observe that this is not equivalent to stripping such properties from Q. For example, consider a query xy for which the property x does not require a classifier. In this case we set $W(X) = 0$, however we still consider for this query the classifier XY (in addition to Y), whose cost may be non-zero (and may be lower than that of Y).

Another characteristic of our problem definition is the requirement to cover all the queries in Q. Alternatively, one could consider a variant of the problem where queries have different weights (e.g. based on their importance) and there is a bound on the how much can be spent on classifiers construction. Then the goal would be to choose a set of classifiers whose cost is within the given budget, such that the sum of weights of all the queries it covers is maximized. We explain in Section 5 why this variant requires different solution techniques than the ones we employ here and leave it for future work.

[2]Positive classification for a conjunction of conditions yields a positive annotation for each of the individual conditions, and otherwise null (unknown) value.

2.2 Related Problems

Finally, we present definitions and results pertaining to related problems, which will also serve us in the theoretical analysis.

Definition 2.1. [50] In the *Weighted Vertex Cover Problem (WVC)* the input is a weighted undirected graph G, and the goal is to select a subset of the vertices of minimal weight which covers all the edges.

Definition 2.2. [8] In the *Max-Flow Problem* the input is a directed graph G, with a source node s and a sink node t, and each edge (u, v) having capacity c_{uv}. The goal is to find a maximum feasible flow. A flow assigns a positive real edge-flow f_{uv} to each edge (u, v). A feasible flow satisfies two conditions: (1) for every edge (u, v): $f_{uv} \leq c_{uv}$; (2) for every vertex not in $\{s, t\}$: the sum of its incoming edge-flows equals the sum of its outgoing edge-flows.

While the *WVC* problem is known to be NP-hard [16], the following result states that the special case of *WVC* over bipartite graphs is reducible to Max-Flow, which is in PTIME.

Theorem 2.3. *[2] The WVC problem over a bipartite graph can be reduced in linear time to the PTIME Max-Flow problem over a bipartite graph.*

We conclude with an overview of results regarding the Weighted Set Cover problem.

Definition 2.4. [50] In the *Weighted Set Cover Problem (WSC)* the input is a collection $\mathcal{S}$ of m sets over a universe $\mathcal{U}$ of n elements. Each set is associated with a cost, and the goal is to find a subset of $\mathcal{S}$ of minimal cost whose union includes all elements. The *degree* Δ of a *WSC* instance is defined as the cardinality of the largest set in $\mathcal{S}$. Moreover, for any instance of *WSC* its *frequency* is defined as the maximal number of sets any element belongs to. In the special case of the *Unweighted Set Cover Problem (SC)*, all set costs equal 1.

Observe that *WVC* is equivalent to *WSC* with $f = 2$, by equating the edges to the elements and the vertices to the sets.

Theorem 2.5. *[11, 16, 30, 49] The SC problem for $f \geq 2$ and $\Delta \geq 3$ is NP-hard, and is hard to approximate beyond a $\min\{(f-1), \ln\Delta\}$ factor unless $P = NP$. Moreover, if the Unique Games Conjecture (UGC) [29] is true, then this inapproximability factor $(f-1)$ increases to f. All the above bounds hold even for the case where all sets are of the same size Δ, or when all elements appear in exactly f sets.*

Note that the above hardness results apply for *WSC* as well, as it has *SC* as a special case. More generally, all lower

and upper bounds on the approximation of *WSC* and *SC* are the same.

A straightforward greedy algorithm for *WSC* is known with a (nearly) tight approximation factor of $(\ln \Delta + 1)$ [6] (observe that $\Delta \leq n$). This algorithm chooses at each iteration the set that maximizes the ratio between the number of elements it covers (not including previously covered elements) and its cost. Its running time is therefore $O(nm)$, which can be improved using priority queues to $O(\log m \cdot \sum_{s \in \mathcal{S}} |s|)$ [9]. Moreover, for a bound dependent on f, a simple polynomial LP-based algorithm [50] achieves an approximation factor of f, which is tight if the UGC is true. By combining these two algorithms, we get the following result.

Theorem 2.6. *[6, 50] The WSC problem admits a PTIME* $\min\{(f, \ln \Delta + 1)\}$*-approximation algorithm.*

3 PREPROCESSING

We next describe our preprocessing pruning procedure, which is the initial step in all our algorithms. This procedure consists of four steps each corresponding to a simple observation regarding the characteristics of an optimal solution.

We show that our pruning preserves the optimal solution (more precisely, at least one optimal solution is preserved), and can only improve the parameters in our results. While theoretically there is no guaranteed worst-case improvement in quality or efficiency, we show in our experiments that in practice there is a significant improvement in both (for $k = 2$ only the running time is improved, since the quality is already optimal).

The complete procedure is depicted in Algorithm 1, and we refer to it in our description below. When analyzing our algorithms in Sections 4 and 5, we assume this processing procedure has already been employed.

Algorithm 1: Preprocessing

Step 1 (Observation 3.1):

1 Select all classifiers for singleton queries
2 Select all classifiers of weight 0
3 Remove all queries covered by the previous steps

Step 2 (Observation 3.2):

4 Build a graph with nodes as properties and a path between any two properties from the same query
5 Discover connected components using a BFS algorithm
6 Partition the problem into independent sub-instances based on the connected components

Step 3 (Observation 3.3):

7 Go over all classifiers by increasing length (2 to k)
8 For each classifier S consider all combinations of two classifiers whose union is S, and for each of the two, if already removed, consider instead the least costly decomposition which was marked to replace it
9 If the least costly of these decompositions of S does not cost more than $W(S)$ then remove S and mark this decomposition to replace it
10 Check for every query if it now has only one cover possibility, and if so select the corresponding classifiers
11 keep repeating Step 3 for all classifiers that intersect with at least one classifier selected in line 10

Step 4 (Observation 3.4): if $k = 2$:

12 For every singleton classifier X, if the overall cost of the set S_X of all classifiers the intersect X satisfies $W(S_X) \leq W(X)$, then select S_X and remove X
13 For every selection of a classifier XY along with a removal of X, recheck this condition for Y

Step 1 - Singleton queries and zero weight classifiers.

Observation 3.1. *For every singleton query the solution must include its corresponding singleton classifier.*

Following this trivial observation, we first select all such classifiers (line 1 in Algorithm 1). Selecting a classifier can be modeled by setting its weight to 0, after adding its weight to the cost of the solution. More generally, we select all classifiers of weight 0 (line 2).

We also remove from the input all queries that are now covered by these selected classifiers, as well as any classifier which is no longer relevant for the remaining queries (line 3). Removing a classifier can be modeled by setting its weight to ∞. This step is linear in the input size.

We can assume henceforth that there are no singleton queries in the input.

Step 2 - Decomposition into smaller problems.

Observation 3.2. *If the query set Q can be partitioned into $m > 1$ subsets $\{Q_1, Q_2, ..., Q_m\}$, such that every two subsets are disjoint in terms of properties, then the optimal solution is a union of the m optimal solutions pertaining to each sub-instance where the query set is restricted to Q_i for $i \in [m]$.*

To discover such a partition into the maximal number of subsets, we use the following algorithm. We build an undirected graph whose nodes are the properties, and for every query we choose some arbitrary order of its properties, and connect every two nodes that correspond to two adjacent properties (line 4).

A simple BFS procedure then yields the connected components (line 5), which in turn induce the partition of the queries (line 6). The number of edges in the graph is bounded by the number of classifiers, hence this algorithm is linear.

This step allows us to solve all sub-instances in parallel. For simplicity, we assume henceforth that the input translates into a single connected component.

Step 3 - Removing classifiers with less costly alternatives.

Observation 3.3. *For any given classifier X and a set $\mathcal{S}$ of classifiers of length at most $|X|-1$, such that $\cup_{S\in\mathcal{S}} = X$, if $W(X) \geq \sum_{S\in\mathcal{S}} W(S)$, then X can be removed.*

This step entails removing classifiers whose range of applicability is subsumed by a set of shorter classifiers of at most the same cost. To illustrate, consider the weights $W(X) = W(Y) = 1$ and $W(XY) = 3$. We can disregard the option of selecting XY, since selecting X and Y instead subsumes the covering contribution of XY and costs less.

Since k is constant, even a naive implementation testing all relevant combinations takes linear $\Theta(n)$ time. Nevertheless, we provide a more efficient implementation, avoiding redundant comparisons (moreover, this procedure is worth running regardless of efficiency, as in our offline setting the most important aspect is the cost of the solution). Concretely, we iterate over all classifiers by increasing length starting from 2 up to k (line 7), and consider for each classifier S all combinations of two classifiers whose union is S (line 8). We call each combination a decomposition of S of size 2. If the combined cost of the least costly decomposition does not exceed $W(S)$, then we remove S from the input (line 9). If one of the two classifiers in the decomposition has been previously removed, then we consider instead its own least costly decomposition. For example, suppose XY was removed and $\{X, Y\}$ has been marked as its least costly decomposition (in this case it is also the only one). Later, when we move on to examining classifiers of length 3, and in particular XYZ, for which we consider several decompositions of size 2, the weight of any decomposition that includes XY would be computed by replacing its weight with the weight of its least costly decomposition $W(X) + W(Y)$.

We next check for every query if there remains only a single way to cover it, due to the removal of classifiers. If so, we then select all these classifiers (line 10).

Finally, we keep repeating this step, each time reexamining only classifiers that intersect with the selected singletons (line 11). Since the number of times we recheck the decompositions of any given classifier is bounded by the number of its sub-classifiers, which is $O(1)$ for constant k, the overall complexity remains $\Theta(n)$.

Step 4 - for $k = 2$: removing singleton classifiers with less costly alternatives.

Observation 3.4. *When $k = 2$, for every singleton classifier X, if the sum of weights of all other classifiers which contain X is at most $W(X)$, then an optimal solution includes all these classifiers and not X.*

We test the above condition for every singleton classifier, and if it holds, we remove it and select all the classifiers that intersect with it (line 12). Here as well, a chain reaction of pruning steps can occur. After removing a classifier X, for each classifier XY selected (its weight is now 0), the pruning condition now may transition from false to true for Y (line 13). As every further pruning action induced involves selecting at least one more classifier which covers an entire query, this whole final pruning step is performed in linear time.

We remark that we employ further optimizations in implementing the above procedure, however the precise description is convoluted, and the high-level ideas and overall $\Theta(n)$ complexity remain the same.

4 QUERIES OF LENGTH TWO

In this section we present the theoretical study of the special case of the MC^3 problem where $k = 2$, before moving on to the general problem in the next section. The reason for studying this variant separately is twofold. First, while we will show that the general problem is NP-hard, this is not the case here. We promptly show that this variant can be solved exactly in polynomial time. Second, as we show in our experiments, in some settings queries of length at most two account for more than 95% of the query load. In such cases we demonstrate that first solving the problem only for these short queries is the best strategy. We now prove that the special case of $k = 2$ reduces to the Max-Flow problem, which is well-known to have efficient algorithms for finding the optimal solution. Moreover, the resulting Max-Flow instance has characteristics, most notably bipartiteness, which allow for further optimizations beyond the general problem.

Theorem 4.1. *The MC^3 problem, where $k \leq 2$, can be solved exactly in PTIME. In particular, its time complexity is upper-bounded by the complexity of solving the Max-Flow problem over bipartite graphs where the number of nodes and edges is $O(n)$.*

Proof. The case of $k = 1$ is trivial since the only solution is to choose all classifiers. Otherwise, given an instance $\langle Q, W\rangle$ of MC^3 where $k = 2$, we reduce it to an instance $\langle G\rangle$ of the Weighted Vertex Cover Problem over a bipartite network, which is known to be in PTIME via a straightforward reduction to the Max-Flow problem (Theorem 2.3).

Concretely, we construct a bipartite graph G whose weighted vertex set is $L \cup R$. The set L contains a node per each singleton in C_Q (denoted the same as the classifier, as a slight abuse of notation), whereas the set R contains a node per each classifier of length 2. In both cases the weight of the node equals the weight of the corresponding classifier (here, for simplicity, we include classifiers with infinite weight in the input). Recall that following our preprocessing there are no singleton queries, hence here all queries are of length 2. We add for each query xy two edges which are adjacent to the node XY: one connecting it to X and the other to Y.

Algorithm 2: MC^3 solver for k=2

1 Run preprocessing procedure (Algorithm 1)
2 Reduce to bipartite WVC (proof of Theorem 2.3)
3 Reduce bipartite WVC to Max-Flow ([2])
4 Run a Max-Flow algorithm [10]
5 Translate to a MC^3 solution (as described in [2])

The nodes represent the corresponding classifiers, the edges represent the corresponding queries for which they were added, and in particular, each single edge represents a property of the query. It is easy to show that the two instances are completely analogous and their solutions are equivalent.

Note that the number of nodes and edges is at most $2n$ each. In the reduction to Max-Flow [2] only two nodes are added (the source and sink nodes), and the number of added edges is at most $2n$ (one per each node). Therefore, the entire reduction is linear in n, and the number of nodes and edges in the resulting graph is also $O(n)$. □

Our full algorithm is depicted in Algorithm 2. It consists of the pruning procedure (line 1) described in Section 3, followed by the reduction to Max-Flow (line 3), via an intermediary reduction to WVC (line 2) as described in the proof above, and a solution of the generated Max-Flow instance (line 4) with translation back to an MC^3 (line 5). As all other steps are linear, the running time of Algorithm 2 is dominated by the Max-Flow algorithm. The (theoretically) best known strongly polynomial algorithm for sparse networks is [40], which runs here in $O(n^2/\log n)$. Since the resulting Max-Flow instance is bipartite, in addition to being sparse, optimized algorithms exist [1]. The choice of the best algorithm depends on several parameters such the as maximum capacity of an edge and the size n_1 of the smaller set of nodes out of L and R. our experimental study led to the choice of the algorithm originally described in [10] which runs here in $O(n_1^2 \cdot n)$. Further discussion appears in our Experiments and Related Work sections.

Almost $k = 2$. As demonstrated in our experiments (Section 6), some domains produce datasets in which a very high percentage of queries (but not all) are of length at most 2. In such cases we suggest the following heuristic: we first cover only these short queries using the optimal Algorithm 2, followed by running our algorithm for the general problem (Algorithm 3 in Section 5) on the residual problem. We report in our experiments that this heuristic is likely to be the best strategy in such cases.

5 GENERAL PROBLEM

In this section we study the general MC^3 problem, where k, the maximal query length, may exceed 2. As mentioned in the preliminaries, while we also provide results for super-constant k, unless stated otherwise, k is assumed to be a constant. We show here that MC^3 is NP-hard, and provide hardness bounds on its approximation, which hold even for a considerably restricted variant. We then present our algorithm, that is based on a reduction to WSC, and comes with approximation guarantees. We conclude this section by discussing how our results apply for several common variations of our model.

Before presenting our results, we first define additional useful notation, along with an *incidence* parameter, which facilitates more granular approximation bounds. For any subset of properties $S \subseteq P$, we denote by $Q_S \subseteq Q$ the set of queries which include S. We also define for a classifier S, if $W(S) \neq \infty$, its incidence $I(S) = |Q_S| \leq n$ as the cardinality of this set. If $W(S) = \infty$, then $I(S) = 0$. Accordingly, we define the incidence $I = \max_{S \in C_Q} I(S)$ of an MC^3 instance, as the highest incidence of all classifiers.

For example, for $Q = \{xy, yz\}$ with all classifier costs being finite, we have $Q_x = Q_{xy} = \{xy\}$, $Q_z = Q_{yz} = \{yz\}$ and $Q_y = \{xy, yz\}$, implying $I(x) = I(z) = I(xy) = I(yz) = 1$, and $I = I(y) = 2$.

5.1 Approximation Hardness

We first prove the hardness of MC^3 in n, the number of queries, which holds even in limited settings and extends to any $k > 2$ which is at most polynomial in n. Then, we also show that MC^3 is NP-hard in k (when there is no restriction on its size) independently of n, once again even in a considerably restricted setting.

Theorem 5.1. *The MC^3 problem is NP-hard for any $k > 2$ where the number of classifiers is polynomial in n, and has no approximation factor below $\min\{(k-2), \ln I\}$ (I is the incidence defined above), unless $P = NP$. Moreover, if the UGC is true, then this bound improves to $\min\{(k-1), \ln I\}$. Finally, these bounds hold even for the special case where all classifiers (of finite weight) are of length 2, all weights are in $\{0, 1\}$, and all queries are of length exactly k.*

Proof. We use here the (unweighted) Set Cover problem (SC) from Definition 2.4. We describe an approximation-preserving reduction from SC with parameters $f > 1$ and $\Delta > 2$, where all elements appear in exactly f sets to MC^3 with parameters $k = (f + 1) > 2$ and $I = \Delta > 2$. Theorem 5.1 then follows from Theorem 2.5. Given an instance $\langle \mathcal{S}, \mathcal{U} \rangle$ of SC, we transform every set $s \in \mathcal{S}$ into a distinct property s, and for each element in $\mathcal{U}$ we add a query to Q whose properties are the sets this element belongs to, along with a special property e (the same e for every element), that appears in every query. All properties except for e are referred to as *set-properties*. The weights of the classifiers are assigned as follows. We assign the weight 0 to all classifiers of length

2 that test two set-properties. We assign the weight 1 to all classifiers of length 2 that test e along with one other set-property. All other possible classifiers (whose length does not equal 2) are not added to C_Q (if classifiers of infinite weight were part of the input, this reduction would be polynomial only for $k = O(lgn)$). We can assume that all queries are distinct, as elements that belong to exactly the same sets can be "merged", and do not add to the complexity of the SC problem. Clearly, the number of queries equals the number of elements (both denoted by n), $I = \Delta$, and $k = (f + 1)$, which is the length of every query. To illustrate this construction, consider an element that appears in two sets, x and y. It follows that we add a query xye, and assign the classifier weights $W(XE) = W(YE) = 1, W(XY) = 0$. Observe that for any query, one can use free queries of length 2 to test all set-properties. Thus, at least one classifier of length 2 that tests e and some set-property must be selected. The cost of any solution therefore is the number of such classifiers selected. Given an α-approximation algorithm for MC^3, which produces a solution T for this instance, we construct a solution T' for the SC instance. Concretely, we examine only classifiers of the form XE in T, and for each such classifier add to T' the corresponding set x. Both solutions have the same cost. Moreover, each classifier XE covers exactly the queries that correspond to the elements the set x covers. It is straightforward to show that this equivalency of solutions works in both directions and always preserves the cost, implying an α-approximation algorithm for SC as well. □

Theorem 5.2. *The MC^3 problem is NP-hard in k, even when $n = 1$ and all (finite) weights equal* 1.

Proof. We reduce an instance $\langle \mathcal{S}, \mathcal{U} \rangle$ of SC to an instance of MC^3 with a single query of length k. Concretely, the query has one property for every element in U, and for every set $S \in \mathcal{S}$ we add a classifier of weight 1, whose properties are the corresponding elements in S. it is straightforward to see that the instances are equivalent. We note that we rely here on the result that in the hard instances of SC, the number of sets is polynomial in the number of elements [49], to ensure a polynomial reduction. □

5.2 Algorithm

We now present an approximation algorithm for MC^3 with $k > 2$. We use here the WSC problem from Definition 2.4. Our algorithm is based on an initial reduction to WSC (following the preprocessing procedure in Section 3). We then use known algorithms for WSC. The rest of this subsection is structured as follows. We first explain the reduction to WSC, then analyze the resulting parameters of the MC^3 instance, followed by an analysis of the approximation guarantees

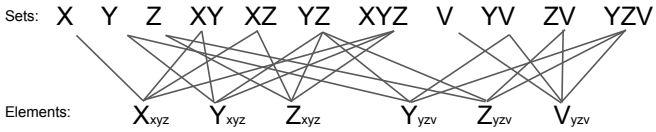


Figure 2: Reduction to WSC.

of two known algorithms for WSC, and finally put everything together into a single approximation algorithm for the general MC^3 problem.

Reduction to WSC. Given an instance $\langle Q, W \rangle$ with a property set P, we reduce it to an instance $\langle \mathcal{S}, \mathcal{U} \rangle$ of WSC as follows. For every query $q \in Q$, for each property $p \in q$ we add the element p_q to $\mathcal{U}$ (note that a distinct element is created for each occurrence of the same property in a different query). The classifiers become the sets in $\mathcal{S}$ (and are denoted the same). The cost of each set equals the weight of the corresponding classifier. An element x_q is added to a set S if and only if both $x \in S$ and $S \subseteq q$ hold. Every (approximated) solution to this WSC instance is translated directly to the same solution for the MC^3 instance (selecting the classifiers corresponding to the sets). One can see that to cover a query of length k', one can choose exactly the same classifiers that correspond to sets whose union contains the k' properties created for this query. Hence, in this reduction as well, the instances are completely analogous, and equivalent in terms of the possible solutions and their costs. To illustrate this reduction, consider an instance of MC^3 where $P = \{x, y, z, v\}$, $Q = \{xyz, yzv\}$ and C_Q contains all possible relevant classifiers, all of weight 1. The resulting WSC instance is then depicted in Figure 2.

Parameter analysis. In the resulting WSC instance the number of elements in $\mathcal{U}$ is the sum of lengths of all queries in Q, and is denoted by $\hat{n} = \sum_{q \in Q} |q| \leq nk$. The number of sets in $\mathcal{S}$ is the same as the number of classifiers, which is denoted by $\hat{m}$, and it follows that $\hat{m} \leq n \cdot (2^{k-1})$ (all queries disjoint and of length k). The frequency pertains to elements corresponding to properties in a query of length k, which in our reduction is $f = 2^{k-1}$ (the number of sets corresponding to the classifiers that test the given property along with any possible subset of the other properties in the query). The size of a set which corresponds to a classifier S is $|S| \cdot I(S)$ (the length times the incidence), since it covers $|S|$ elements for every query it appears in. While all queries are distinct, it is possible that a subset of size $(k-1)$ appears in I queries, hence the degree is bounded by $\Delta \leq (I \cdot (k-1))$.

Approximation guarantees of WSC algorithms. It follows from Theorem 2.6 that the greedy [6] and the LP-based [50] algorithms for WSC yield, respectively, the approximation guarantees $(\ln(\min\{\Delta\}) + 1)$ and f. By assuming the worst

Algorithm 3: MC^3 solver for general problem

1 Run preprocessing procedure (Algorithm 1)
2 Reduce to WSC
3 Run the greedy algorithm for WSC [6]
4 Run the LP-based algorithm for WSC [50]
5 Select the least costly of the two outputs and return the corresponding classifiers

case values for these parameters in our setting, and combining these two algorithms, we get an approximation guarantee of $\min\{\ln I + \ln(k-1) + 1, 2^{k-1}\}$.

The reduction to WSC followed by this compound algorithm is the core of our solution approach, along with the preprocessing procedure. Note that the 2^{k-1} factor, while exponential in k, for small values of k which are relevant in practice, is likely to be the dominating guarantee.

Putting it all together. Our full algorithm is depicted in Algorithm 3. It first runs our preprocessing procedure (line 1), followed by the reduction to WSC (line 2), and then, finally, runs both the greedy algorithm or the LP-based algorithm for WSC (lines 3 and 4). The output is the set of classifiers corresponding to the collection of sets produced by the WSC algorithm with the least costly output (line 5).

As the preprocessing procedure does not affect the conditions necessary for the approximation guarantees to hold, we have the following theorem.

Theorem 5.3. *Algorithm 3 provides an approximation guarantee of* $\min\{\ln I + \ln(k-1) + 1, 2^{k-1}\}$ *for the* MC^3 *problem.*

We remark that there are other approximation bounds for WSC that combine the frequency and degree parameters (see, e.g., [42]), however, as somewhat improved bounds are possible in only very specific parameter ranges, such level of granularity is arguably not pragmatic to discuss here, and hence omitted.

The time complexity of Algorithm 3 is dominated by the LP-solver, which is in the worst case polynomial, and as we show in our experiments runs in reasonable time. All other steps have running time $O(n)$ for constant k, following the complexity discussions in Sections 2 and 3 along with the linear complexity of the reduction to WSC. We note that while running both WSC algorithms, instead of only the one with the better worst-case guarantees, we sacrifice running time for potentially improved quality, which in our motivating setting is more important.

5.3 Special Cases and Generalizations

We now consider important extensions of our model, which capture common real-world settings. We explain how we adapt our approach to solve these variations (and analyze how we can potentially even improve our bounds), or, for a specific generalization, demonstrate why this cannot be done, requiring a different approach altogether. Finally, note that the special case where almost all queries are of length at most 2 is discussed in Section 4.

Bounded classifiers. As mentioned in Section 2, in practice when the number of theoretically possible classifiers is too large, some are not considered. In particular, a prevalent approach is to consider only classifiers of length at most $k' < k$, which is often $k' = 2$. While Theorem 5.1 proves that our hardness results hold even when all classifiers are of length exactly 2, we can nevertheless ameliorate the gap between these bounds and our approximation guarantees, as in our reduction to WSC we get improved parameters. Specifically, the frequency becomes at most $f \leq \sum_{i=0}^{k'-1} \binom{k-1}{i} = 2^{k'-1}$ (a classifier must include the property which corresponds to the element, along with any subset of size at most $(k'-1)$ of the remaining $(k-1)$ properties in the query), which in general can be approximated using various bounds for sums of binomial coefficients for specific ranges of k' and k. Importantly, for $k' = 2$ we get $f \leq k$. Note that for $k' = k$ this sum is well-known to equal 2^{k-1} (which is consistent with our analysis for the general case). As for the degree, it is now bounded by $\Delta \leq (k'-1) \cdot I$ (compared to $(k-1) \cdot I$ in general), following the same reasoning as in the analysis of the general case.

Multi-valued classifiers. As mentioned in the Introduction, while binary classifiers are often preferred in practice for higher precision, our model can be extended to represent multi-valued classifiers as well. A multi-valued classifiers acts as the union of all binary classifiers for properties which pertain to values of the same attribute. For example, consider the properties x and y where x is "color = red" and y is "color = blue". We can merge the two properties into a "color" property c and consider a dedicated multi-valued classifier C for c. Such a classifier can determine the color of any item, and therefore acts as a binary classifier for any color property, in particular x and y.

If we consider in our model, instead of binary classifiers, only multi-valued classifiers, then by merging all properties belonging to the same attribute, we end up with the same abstract MC^3 problem (even though any concrete instance is reduced in input size following the transformation). For instance, continuing with Example 1.1, suppose we have the same two search queries, where the first asks for a "team = Juventus, color = white, brand = Adidas" soccer shirt (q_1), and the second for a "team = Chelsea, brand = Adidas" soccer shirt (q_2). These queries contain 4 properties corresponding to 3 attributes: team, color and brand (note that the attributes induce an equivalence relation over the properties). Therefore, in a setting with only multi-valued classifiers, the properties

Table 1: The datasets used in the experiments

Dataset	# of queries	Max cost	Max length
BestBuy (BB)	1000	1	4
Private (P)	10,000	63	5
Synthetic (S)	100,000	50	10

are now the attributes b (brand), c (color) and t (Team), with the queries being $q_1 = tcb$ and $q_2 = tb$ (the original queries are of course also retained, but are omitted from the abstract model). The classifiers are B, C and T (for example, T can determine whether the team associated with a given soccer shirt is "Juventus", "Chelsea", or any other team relevant to the entire query load, if more queries are provided). The weights of these classifiers are provided based on external estimations of their training cost. We, thus, end up transforming the original MC^3 instance into a different MC^3 instance, adhering to exactly the same model.

Furthermore, one can also consider multi-valued classifiers alongside binary classifiers. The multi-valued classifiers here make sense only when their cost is less than the sum of costs of the corresponding binary classifiers (as otherwise they are not necessary for an optimal solution and can be pruned). We can extend our reduction to WSC by adding sets to represent these classifiers, which then cover the corresponding elements (that correspond to properties pertaining to different values of the same attribute for which this classifier pertains). For example, continuing with the soccer shirts example, a classifier T (team) would transform into a set T which covers elements corresponding to the c (Chelsea) and j (Juventus) properties. The analysis of this extended WSC instance then follows along the same lines as the analysis for the strictly binary classifiers.

Partial cover. As mentioned in the introduction, a possible variant of our problem is one where weights are assigned to queries, reflecting their importance, and the goal is to find a solution which maximizes the combined weight of the covered queries, given some budget constraint on the overall costs of classifiers. Observe that our reduction to WSC no longer works for this variant. While the reduction to WSC creates an equivalent instance to the original MC^3 instance, when one asks for a complete cover, this equivalency does not extend to partial covers. For example, for a single query xyz, covering only the elements x and y increases the cover in the WSC setting, however does not cover the original query (research shows that only partially conforming to search criteria can in some cases even have a worse effect on user satisfaction than not conforming at all [23]). We, therefore leave this variant for future work, and only note here that, in fact, one can show that this problem is much harder to approximate.

6 EXPERIMENTAL STUDY

In this section we present the experimental evaluation performed on various datasets, including synthetic and real world data.

6.1 Experimental Setup

Our algorithms were implemented using Python, and we ran the experiments on a server with 128GB RAM and 32 cores. To evaluate our solution, we have performed a set of extensive experiments on a publicly available real-world dataset, a larger (private) dataset provided by a large e-commerce company[3] (which consists of several datasets pertaining to different product categories), and a synthetically generated dataset.

Datasets. As noted above, we performed our evaluation on three datasets. First, we used a small, publicly available, dataset from BestBuy, which had been used by [13] for their evaluation. The dataset consists of about 1000 queries from the electronics domain. It has uniform weights, and 95% of its queries have up to 2 properties specified. The second dataset is private and comes from a large e-commerce company. It consists of 10,000 popular queries of various lengths (1 to 6 properties specified) and varying costs for classifiers (1 to 63). These costs represent a normalization by a factor of N, which is an internal measure of the e-commerce company, of the estimated number of labeled examples experts must annotate in order to train the corresponding classifier to the required precision. Importantly, the monetary cost is linearly correlated with these specified costs (i.e. annotating one example costs a fixed amount of dollars). This dataset is in fact a union of several sub-datasets pertaining to different categories of products (Electronics, Fashion, Home & Garden). We note that in the fashion category there are roughly 1000 queries, 96% of which are of size at most 2, and for this quality we run separate experiments on it as well. The third dataset is generated synthetically, and consists of $n = 100,000$ queries. The costs are drawn from a uniform distribution over the range $[1, 50]$. The length of any generated query equals $l > 1$ with a probability $\frac{1}{2^{l-1}}$, i.e. half of the queries are of length two, quarter of the queries are of length three, and so on. This corresponds to the common real life inverse correlation of queries length and frequency. Queries generated with length exceeding 10 are omitted, because such long queries are rare in practice [21] (hence companies do not invest money in corresponding classifiers). Into each query we select properties uniformly from a pool of n/t properties, when t is uniformly selected to be in $[2, \sqrt{n}]$. This dataset is regenerated for each separate experiment. Table 1 depicts a summary of all the datasets.

[3]Company name omitted due to privacy considerations.

Algorithms. We compare 4 different algorithms for short queries (up to two properties) and 5 different algorithms for general length queries. The experiments are performed separately for each of these two problem settings.
We first present the following algorithms we have tested for short queries:

- MC^3[S] - Our main proposed algorithm for $k = 2$ (Algorithm 2).
- **Property-Oriented** - This algorithm selects all the singletons classifiers (and nothing else).
- **Query-Oriented** - This algorithm selects all the classifiers that correspond to entire queries (i.e. one classifier per each query).
- **Mixed** - The algorithm proposed in [13], for the case with uniform costs, which we accordingly use only for the *BB* dataset.

As discussed in Section 4, our proposed algorithm uses as a sub-component an algorithm for the Max-Flow problem over a bipartite graph. We have tested the algorithms in [1] which focused on optimizations for bipartite graphs (mostly improved analysis for already existing algorithms). As all examined algorithms turned out to be highly scalable, we omit here a detailed description of the small performance differences we have observed. We report the best performance was consistently achieved by [10], which is the algorithm we have chosen in all other experiments as part of Algorithm 2.

The following are the algorithms for the general case:

- MC^3[G] - Our proposed algorithm for the general case (Algorithm 3).
- **Short-First** - This algorithm (which we have mentioned in Section 4) combines all our algorithms. It first considers only the queries of size at most 2 and runs Algorithm 2 over these queries, and then runs Algorithm 3 over the residual problem (covering longer queries).
- **Property-Oriented** - This algorithm selects all the singletons classifiers (and nothing else).
- **Query-Oriented** - This algorithm selects all the classifiers that correspond to entire queries (i.e. one classifier per each query).
- **Local-Greedy** - This iterative greedy algorithm finds the least costly cover (taking into account previous selections) of a single query over all queries, and selects the classifiers in that cover. I.e. for each query it inspects all cover options, (there are $O(1)$ such options for constant k). After finding the minimal cover of each query, it chooses the minimum of these minimums, thus covering (at least) one query at each iteration.

Evaluation Outline. We have compared the algorithms listed above over our datasets, both in terms of the overall cost of the selected classifiers and the overall execution time. In addition, we have examined the effect of our preprocessing step (Algorithm 1, described in Section 3) both on the running time of the algorithms and the solution quality. To better capture practical settings where the size of the query load varies according to different budget quotas, for each inspected dataset, along with running the experiments on its entire query load, we also randomly select subsets of this query set of different cardinalities and run the algorithms over these corresponding sub-instances. Accordingly in Figure 3 the x axis displays these varying cardinalities of different query subsets.

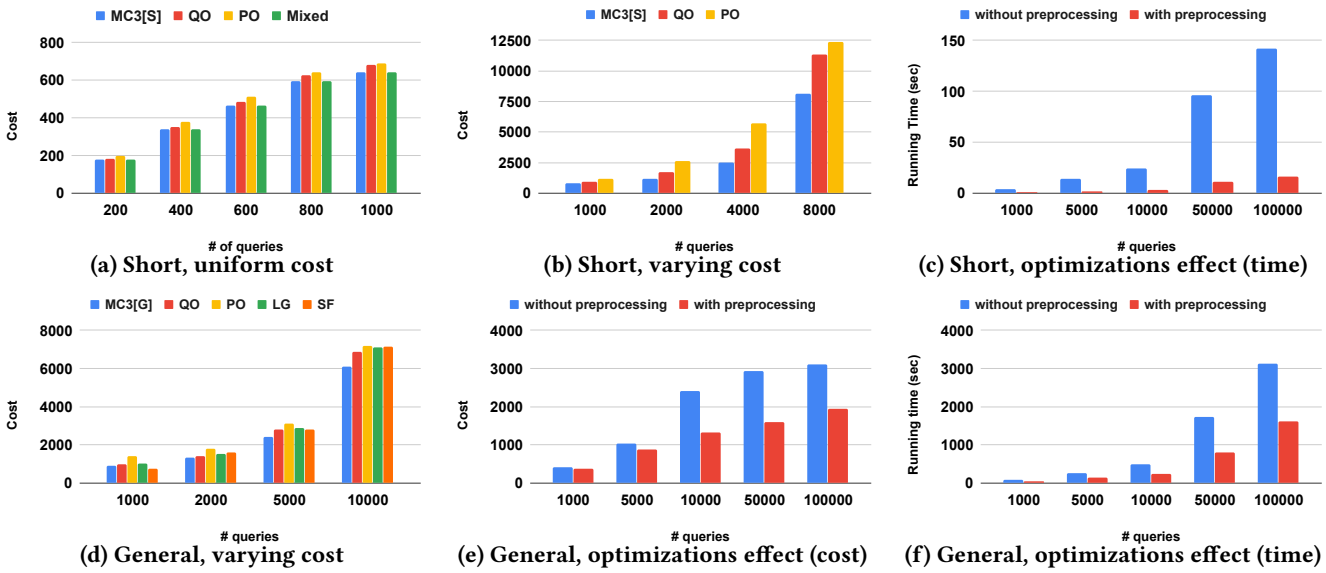
6.2 Evaluation Results

We next present our experimental study results. We start with the results for the short queries (up to two properties).

Comparison to previous work. The Mixed algorithm introduced in [13] works on short queries with uniform classifiers cost. Due to these restrictions, we compare it to our algorithm, and the other baselines, over the *BB* dataset. The Property-Oriented and Query-Oriented competitors also proposed in [13]. Figure 3a depicts the classifiers construction costs achieved by all the competitors. Our algorithm and the Mixed algorithm are both optimal, afterwards comes the Query-Oriented algorithm and finally Property-Oriented algorithm. This experiment demonstrates applicability of our approach to the restricted version of the problem, subsuming the applicability of our predecessor.

Short queries with varying cost. Next, we evaluate the construction costs of our approach compared to all applicable baselines over the *P* dataset, restricted to only short queries (80% of the data). Since this dataset includes varying costs, the Mixed algorithm is not applicable in this experiment. Figure 3b depicts the solution costs achieved by all the competitors. As expected MC^3[S] achieves the optimal results and outperforms both Query-Oriented and Property-Oriented algorithms by 30%, demonstrating that naive/simple solutions are far from optimal.

Running time & Preprocessing effect. Finally, we focus on MC^3[S], the best performing algorithm cost-wise, and examine its running time, and the effect of the preprocessing on the running time. This evaluation is performed on the synthetically generated dataset. The preprocessing step has no effect on the optimal solution cost, hence we do not present it. Figure 3c depicts the running time before and after applying the preprocessing. The running time for the algorithm varies from 4 seconds for 1000 queries to 142 seconds for 100,000 queries. The preprocessing step saves 85% of the running time, yielding an execution time of 20 seconds for 100,000 queries. These fast running times demonstrate the scalability



(a) Short, uniform cost (b) Short, varying cost (c) Short, optimizations effect (time)

(d) General, varying cost (e) General, optimizations effect (cost) (f) General, optimizations effect (time)

Figure 3: Experimental results

of our algorithm. The running time on real-life datasets was similar to that of the synthetic of similar size, hence omitted.

Next, we present the results for the general MC^3 problem, where the length of the queries may exceed two.

Solution quality. We evaluate the construction costs of the classifiers selected by our algorithm for the general case compared to that of all the specified baselines, over the P dataset. For the smallest subset of the P dataset consisting of 1000 queries, SF is the best performing and MC^3[G] is the second best. This subset includes only queries pertaining to the fashion category (and not a random subset of queries). As mentioned, 96% of its queries are of size at most 2, hence it is not surprising that our secondary algorithm, SF, is the best choice in this setting. MC^3[G] achieves the best results over all of the other dataset sizes (recall that we randomly select different subsets of varying cardinalities of the entire dataset, and run the algorithm over these restricted subsets, which of course also includes the entire dataset) and outperforms its closest competitor algorithm by 12%.

Running time & Preprocessing effect - the general case. Next, we present the running time of the MC^3[G] algorithm and the effect of the preprocessing both on the running time and the solution cost. This evaluation is performed over the synthetically generated dataset, which is much larger. Figure 3e depicts the construction cost before and after applying the preprocessing. The preprocessing saves 35% of construction cost. Figure 3f depicts the running time before and after applying the preprocessing. The running time for the algorithm varies from 89 seconds for 1000 queries to 3135 seconds for 100,000 queries. The preprocessing step saves 50% of the running time, yielding an execution time of 1620 seconds for 100,000 queries. Both the running times and the construction costs over the real-life datasets were similar to those over the synthetic datasets of the same size, and hence omitted.

7 RELATED WORK

Supervised machine learning is a key solution for many real world problems, from detecting spam to facilitating autonomous vehicles. Humans are widely employed for various tasks to support supervised ML models, e.g., feature selection [38], learning of semantic attributes [48], image tagging [47], and in many modern hybrid systems [5, 27, 43, 46]. As the human component is the most costly and error prone, many works studied minimization of interactions with human workers, to reduce the error rate or the overall cost [52]. Our current work falls under latter category. Previous work [13] considered a simplified model where all classifiers had the same cost and the queries included at most two properties. We remove these restrictions and prove the generalized model to principally differ from its restricted predecessor in terms of computational complexity and required algorithmic solutions.

A problem close to ours in spirit is the Minimum Substring Cover problem (MSC) [4, 24], where, in high-level, given a set of strings $\mathcal{S}$, one needs to find a subset of strings S of minimal cost, such that all strings in $\mathcal{S}$ can be constructed by concatenating strings in S. There are, however, several crucial differences to our model, which result in a much different theoretical problem. Most notably, we combine classifiers by logical conjunction as opposed to the string concatenation.

This distinction (along with other distinctions to different variations of MSC) leads to entirely different algorithms for MC^3, and different technical characteristics.

Another related line of work is materialized view selection in data warehouses (MVS) [20, 32, 35], where one materializes a set of queries that enable to answer an expected query workload. In such a scenario however the materialized objects are relations, rather than binary classifiers, the query language is richer, including, e.g. joins, and an important objective is optimizing the query workload execution time. Partial query answers are also often tolerable [3], in contrast to our setting where each query must be covered. Similar to our setting, MVS problems can be modeled as cover problems, however the combinatorial characteristics of the covering patterns often greatly differ. For instance, the notable work [22] examines a problem where queries are covered by a single covering element (view) unlike in our setting where in most cases, multiple covering elements (classifiers) are combined to cover a query. As shown in [28] the MVS problem studied in [22], and consequently its various extensions (e.g., [19]) are NP-hard to approximate below a $n^{1-o(1)}$ factor. To our knowledge, neither our algorithms nor our approximation guarantees can be derived from any previous MVS results for models that generalize ours.

Our hardness bounds and algorithms for the general MC^3 problem (Section 5) are based on reductions to and from the classical Weighted Set Cover problem (WSC). There is a large body of work on this problem and its extensions [7, 9, 11, 12, 30, 49]. All results which we have relied on in our work are described in Section 2. Our lower and upper approximation bounds correspond to WSC bounds, that are functions of the frequency and degree parameters separately. By examining additional bounds that combine frequency and degree in the same factors [14, 39, 42] one gets additional bounds for our problem based on the same reductions. The same applies to works examining the hardness of WSC based on the number of sets [37]. It is important to note that WSC with frequency f is equivalent to the Weighted Vertex Cover problem (WVC) over hyper-graphs where hyper-edges are of size at most f. Many results for WSC are, in fact, studied via the hypergraph WSC formulation [11, 15, 39]. Due to the similarity demonstrated in our reduction between WSC and MC^3, generalizations of WSC may potentially aid in extending our model. These problems include the Set/MultiSet MultiCover problems [25, 26, 41], where there are various restrictions on the number of times each element can be covered, and multisets are allowed.

Our exact solution for queries of size of at most 2 (Section 4) is based on a reduction from a special (PTIME) case of WVC to the Max-Flow problem. This reduction is considered folklore, and is described, e.g., in [2]. The Max-Flow problem has been extensively studied for decades, and many efficient algorithms have been devised for it [17, 18, 31, 34]. In particular, our reduction results in a sparse graph, and for this case [40] presented the best-known strongly polynomial algorithm. There also exist incomparable algorithms [34], whose complexity is based on the maximum capacity of an edge. For practical running time, the algorithms with the best theoretical complexity are often inferior to simpler algorithms [33, 36, 51]. Optimized algorithms for bipartite graphs have been studied in [1], and empirically compared in [36]. In our experiments we employed the algorithms in [1], with the choice of the concrete algorithm depending on various problem parameters, as discussed in Section 6.

8 CONCLUSION AND FUTURE WORK

This paper introduced the MC^3 problem, which aims to devise effective algorithms to choose which classifiers to train to address a given search query load, while minimizing the training cost. Importantly, we extended the limited model in [13], by allowing varying costs for different classifiers, and also queries of arbitrary length, and presented novel solution techniques. For $k = 2$ we provided an exact algorithm. Whereas, for the general problem, we proved its approximation hardness in several respects, and provided an algorithm with approximation guarantees. Moreover, our bounds use granular parameters, capturing more precisely what can be done in a practical setting. Our experiments were performed over real-world and synthetic datasets, and demonstrate the efficiency and effectiveness of our algorithms. In particular, we devised a preprocessing procedure, which we have shown to improve running time and solution quality.

Our model assumes that the construction costs of classifiers are independent (see discussion of model characteristics in Section 2.1). Therefore, a natural direction for future work is considering a more general model, where there may be some overlap in the work required for construction of different classifiers. This leads to a more complex cost model, where the cost of constructing specific sets of classifiers may be lower than the sum of their individual costs. Similarly, an interesting problem would be to introduce into the model accuracy levels of classification tasks, and integrate the tradeoff between construction cost and classification accuracy into the problem setting. In our work the cost of each classifier is fixed to match a predefined (implicit) accuracy threshold.

As mentioned at the end of Section 5, we are currently pursuing work on a generalization of our problem to maximizing partial covers under budget constrains, which we prove to be harder to approximate. Finally, we are working on closing the gap between our hardness bounds and our approximation guarantees.

Acknowledgements This work has been partially funded by the Israel Innovation Authority, the Israel Science Foundation, the Binational US-Israel Science foundation.

REFERENCES

[1] R. K. Ahuja, J. B. Orlin, C. Stein, and R. E. Tarjan. Improved algorithms for bipartite network flow. *SIAM Journal on Computing*, 23(5):906–933, 1994.

[2] M. Baïou and F. Barahona. Stackelberg bipartite vertex cover and the preflow algorithm. *Algorithmica*, 74(3):1174–1183, 2016.

[3] J. R. Bernardino, P. S. Furtado, and H. C. Madeira. Approximate query answering using data warehouse striping. *Journal of Intelligent Information Systems*, 19(2):145–167, 2002.

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

[5] J. Cheng and M. S. Bernstein. Flock: Hybrid crowd-machine learning classifiers. In *CSCW*, pages 600–611. ACM, 2015.

[6] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[7] S. A. Cook. An overview of computational complexity. *Communications of the ACM*, 26(6):400–408, 1983.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

[9] G. Cormode, H. Karloff, and A. Wirth. Set cover algorithms for very large datasets. In *CIKM*, pages 479–488. ACM, 2010.

[10] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, volume 11, pages 1277–1280, 1970.

[11] I. Dinur, V. Guruswami, S. Khot, and O. Regev. A new multilayered pcp and the hardness of hypergraph vertex cover. *SIAM Journal on Computing*, 34(5):1129–1146, 2005.

[12] I. Dinur and D. Steurer. Analytical approach to parallel repetition. In *STOC*, pages 624–633, 2014.

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

[14] M. El Ouali, H. Fohlin, and A. Srivastav. A randomised approximation algorithm for the hitting set problem. *TCS*, 555:23–34, 2014.

[15] U. Feige. Vertex cover is hardest to approximate on regular graphs. *Technical Report MCS03–15*, 2003.

[16] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *STOC*, pages 47–63, 1974.

[17] A. V. Goldberg, S. Rao, and S. Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45(5):783–797, 1998.

[18] X. Gong and J. B. Orlin. A fast max flow algorithm. 2018.

[19] 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.

[20] H. Gupta and I. S. Mumick. Selection of views to materialize in a data warehouse. *TKDE*, 17(1):24–43, 2005.

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

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

[23] 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.

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

[25] Q.-S. Hua, Y. Wang, D. Yu, and F. C. Lau. Dynamic programming based algorithms for set multicover and multiset multicover problems. *TCS*, 411(26-28):2467–2474, 2010.

[26] Q.-S. Hua, D. Yu, F. C. Lau, and Y. Wang. Exact algorithms for set multicover and multiset multicover problems. In *ISAAC*, pages 34–44, 2009.

[27] A. Jarovsky, T. Milo, S. Novgorodov, and W. Tan. GOLDRUSH: rule sharing system for fraud detection. *PVLDB*, 11(12), 2018.

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

[29] S. Khot. On the power of unique 2-prover 1-round games. In *STOC*, pages 767–775, 2002.

[30] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2- ε. *JCSS*, 74(3):335–349, 2008.

[31] V. King, S. Rao, and R. Tarjan. A faster deterministic maximum flow algorithm. *J. of Algorithms*, 17(3):447–474, 1994.

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

[33] J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640. ACM, 2010.

[34] A. Madry. Computing maximum flow with augmenting electrical flows. In *FOCS*, pages 593–602, 2016.

[35] 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.

[36] C. S. Negrucşeri, M. B. Pacsoşi, B. Stanley, C. Stein, and C. G. Strat. Solving maximum flow problems on real-world bipartite graphs. *JEA*, 16:3–5, 2011.

[37] J. Nelson. A note on set cover inapproximability independent of universe size. In *Electronic Colloquium on Computational Complexity*. Hasso-Plattner-Institut, 2007.

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

[39] M. Okun. On approximation of the vertex cover problem in hypergraphs. *Discrete Optimization*, 2(1):101–111, 2005.

[40] J. B. Orlin. Max flows in o (nm) time, or better. In *STOC*, pages 765–774, 2013.

[41] S. Rajagopalan and V. V. Vazirani. Primal-dual rnc approximation algorithms for (multi)-set (multi)-cover and covering integer programs. In *FOCS*, pages 322–331, 1993.

[42] R. Saket and M. Sviridenko. New and improved bounds for the minimum set cover problem. In *Approximation, randomization, and combinatorial optimization. Algorithms and techniques*, pages 288–300. 2012.

[43] A. D. Sarma, A. Jain, A. Nandi, A. Parameswaran, and J. Widom. Surpassing humans and computers with jellybean: Crowd-vision-hybrid counting algorithms. In *HCOMP 2015*, 2015.

[44] V. S. Sheng and C. X. Ling. Thresholding for making classifiers cost-sensitive. 2006.

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

[46] 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.

[47] 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.

[48] T. Tian, N. Chen, and J. Zhu. Learning attributes from the crowdsourced relative labels. In *AAAI*, volume 1, page 2, 2017.

[49] L. Trevisan. Non-approximability results for optimization problems on bounded degree instances. In *STOC*, pages 453–461, 2001.

[50] V. V. Vazirani. *Approximation algorithms*. Springer, 2013.

[51] T. Verma and D. Batra. Maxflow revisited: An empirical comparison of maxflow algorithms for dense vision problems. In *BMVC*, pages 1–12, 2012.

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