Synthesizing View Definitions from Data[*]

Anish Das Sarma
Yahoo Research
anishdas@yahoo-inc.com

Aditya Parameswaran
Stanford University
adityagp@cs.stanford.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Jennifer Widom
Stanford University
widom@cs.stanford.edu

ABSTRACT

Given a database instance and a corresponding view instance, we address the *view definitions problem* (VDP): Find the most *succinct* and *accurate* view definition, when the view query is restricted to a specific family of queries. We study the tradeoffs among succintness, level of approximation, and the family of queries through algorithms and complexity results. For each family of queries, we address three variants of the VDP: (1) Does there exist an *exact* view definition, and if so find it. (2) Find the *best* view definition, i.e., one as close to the input view instance as possible, and as succinct as possible. (3) Find an *approximate* view definition that satisfies an input approximation threshold, and is as succinct as possible.

Categories and Subject Descriptors

H.2.m [**Database Management**]: Miscellaneous—*Database Theory*

General Terms

Algorithms, Theory

Keywords

View Definitions, Query Synthesis, Algorithms, Complexity

1. INTRODUCTION

Consider the *View Definitions Problem (VDP)*: Given a database D, and a view V over it, find a view definition Q (i.e., a database query) that best captures the relationship between V and D. Intuitively, we want the view definition Q to be as general as possible, while at the same time, on applying Q to the database D, we want a result V' that is very close to V. VDP is similar to the Query By Output (QBO) problem studied in [15], however our objectives and solutions are different, as discussed in Section 1.1.

[*]This work was supported by the National Science Foundation under grants IIS-0414762 and IIS-0904497.

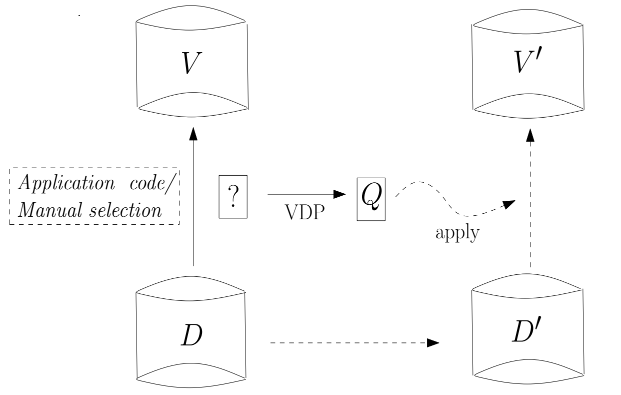

Figure 1: Synthesizing Q from $D + V$ useful in applying Q to different database (or later version) D'.

Considering the triplet (D, V, Q), our goal is to use D and V as input, and obtain Q as the output. From a theoretical standpoint, it is interesting to note that while the other two input/output combinations from this triplet have been studied extensively in the database literature, VDP has not received commensurate attention. Specifically, query processing and optimization address the problem of taking D and Q as input, and finding V as efficiently as possible [6]. The field of answering queries using views [9], and more broadly data integration [8], studies several versions of problems that involve V and Q as input, and seek to arrive at D. For instance, *local-as-view* data integration attempts to obtain a mediated database D from individual sources V and schema mappings Q.

Aside from theoretical interest, there are practical motivations for VDP. To illustrate, consider Figure 1, which shows our given database D and view V on the left. The view V may have been derived manually, e.g., a person selected tuples from D of interest to them, or perhaps V represents the customers that responded to a survey. View V may have also been derived by an application program, but we no longer can or wish to use that program (e.g., the program is hard to maintain, has been lost, or no longer runs on current hardware).

Our goal is to find a query Q that captures what criteria were used to produce V from D. Capturing these criteria can be useful for contextual schema matching [2, 3, 13]. Another reason for wanting Q is so that it can be applied to another database D', as shown on the right of Figure 1. D' could be another database similar to D, or an updated version of D.

(In the latter case, Q is effectively being used for *materialized view maintenance* [7].) By using Q on D' we avoid relying on humans or hard-to-maintain code. Furthermore, if our needs change, it may be much easier to change a query than to change code or to change human procedures.

We now give an example to illustrate the issues we need to deal with to solve VDP:

EXAMPLE 1.1. *Consider a view $V(A, B)$ with n tuples and attribute A as the primary key. Let the database D consist of the relation $R(A, B)$, with the same schema as V. Let the values of the attribute A be $a_1, a_2, \ldots, a_n$ for the n tuples in V. We could construct a view definition Q: "select * from R where $A = a_1$ or $A = a_2$ or ... or $A = a_n$." However, if V is also derived by selecting all and only tuples from R that have $B > 5$, then we could also construct Q': "select * from R where $B > 5$." Both Q and Q' exactly produce V, but Q' is a much more succinct query. Even if Q' only approximately captures V (with a few additional tuples, or a few tuples missing), we might still prefer Q' over Q, not just because it is succinct, but also because it has a single predicate, and therefore may be more efficient to evaluate.*

In fact, there are three factors that interact closely while finding a solution to VDP:

1. *Family of Queries:* Sometimes it is necessary to restrict Q to be from a specific family of queries, e.g., single predicates or conjunctive queries. The database D may only support certain types of queries, or the family of queries may guarantee efficient evaluation. Additionally, the family of queries can impact the performance of finding a solution to VDP.
2. *Level of Approximation:* We may return an *approximate* solution Q for VDP, i.e., Q applied to D produces a V' that is close to but not exactly V, for three reasons: (1) There may not be a query Q in the family of queries considered that exactly produces V. (2) Allowing an approximate solution may lead to a more efficient procedure for finding Q. (3) The approximate query may be more succinct, as we discuss next.
3. *Succinctness:* We would like our queries to be "succinct." In the example above, we would consider Q' to be considerably more succinct than Q, since it applies many fewer conditions to obtain its result. A more succinct query is typically more readable, understandable, and maintainable. Moreover, as we've seen in the example above, a more succinct query is usually more *general*, in that it can be applied meaningfully to other databases similar to D. Typically, the fewer conditions query Q applies to obtain its result, the less specific it is to the given database D.

   Note there is a metric related to succinctness that we will call *experimental generality*, or generality for short. Generality is evaluated on test data sets, separate from D and V, and captures how well Q predicts views in the test data set. Unlike generality, succinctness is an intrinsic property of a query. We believe that a succinct query will typically be more general than a less succinct query, as the extra conditions in a less succinct query tend to overfit the data. In addition, succinctness is also desirable because it yields more readable (and more maintainable) queries. We discuss experimental generality, and its relationship to succinctness, in more detail in Section 1.1.

We formalize our notions of succinctness and approximation in subsequent sections, and explore the connections between them for several query families. As we will see, in each query family there is a tradeoff between succinctness and level of approximation. At the extremes, if Q =*false*, i.e., Q does not select any tuples, then it is very approximate (it misses all the tuples in V), but is extremely succinct. On the other hand, if Q has a condition that selects each tuple appearing in V and no others, then Q is exact, but not succinct.

In this paper, we study three VDP subproblems: the *Exact View Definition (EVD)*, the *Best View Definition (BVD)* and the *Approximate View Definition (AVD)* problems. We study the complexity of these three subproblems for each of our query families. For the cases where a given subproblem is intractable, we identify the complexity and provide algorithms giving approximate solutions. In other cases, we provide polynomial time optimal solutions.

The paper is organized as follows. Section 2 formally defines the problems we consider. In Sections 3–6, we address the problems for each family of queries. Related work is presented next (Section 1.1) and we conclude with future work in Section 7.

1.1 Related Work

Our work can be differentiated from related work in two ways: the scope of the problem addressed, and the evaluation metrics.

Related to scope, we address a narrower problem than others but with more depth. For example, *decision trees* [12] are used to classify items. Our VDP can be viewed as a classification problem with only two classes: tuples in V and those not in V. Thus, VDP is narrower since only two classes are considered. However, we go deeper into this subproblem by considering multiple query families and multiple optimization criteria. Similarly, Query By Output (QBO) [15] is a more general problem than ours. In QBO, we are given V and $R_1, R_2, \ldots, R_n$, and asked to return Q such that $Q(R_1, R_2, \ldots, R_n) = V$. Thus, VDP is the special case where $n = 1$ and there are no joins nor projections. However, as mentioned above, we thoroughly study the VDP subproblem by considering different classes of selection queries and multiple optimization criteria.

The second differentiator for our work is the metrics we use: succinctness and level of approximation. Most classification work, on the other hand, relies on experimental generality, mentioned earlier. When generality is a goal, we are given, in addition to D and V (the training sets), two test data sets, D_T and V_T. Query Q is determined without seeing the test sets, only using the training sets. However, Q is evaluated by the level of approximation of $Q(D_T)$ to V_T. Generality is an experimental measure, although one can study generality by assuming that D and D_T (and V and V_T) have "similar" statistical properties. On the other hand, our notion of succinctness is an intrinsic property of a query. Intuitively, a succinct query will be more general than a less succinct query that overfits the data. In addition, succinctness is also desirable because it yields more readable (and more maintainable) queries. When succinctness is a goal (as it is in this paper), we can obtain theoretical guarantees, e.g., an algorithm can find the most succinct Q within a family of queries in quadratic time.

Next we present some additional details about decision trees and QBO.

Classification: Decision trees are used to classify items into $n \geq 2$ classes. A decision tree is simply a binary tree with each tree node annotated with a predicate. One child of the node corresponds to the predicate evaluating to true, and the other child corresponds to the predicate evaluating to false. Each leaf node is labeled with a class label from $C_1, \ldots, C_n$. For VDP, we restrict $n = 2$ (tuple present in the view, or not present in the view).

Thus a decision tree for $n = 2$ corresponds to a query of the following form: $Q = (p_1 \wedge Q') \vee (\neg p_1 \wedge Q'')$, where p_1 is the splitting predicate, and Q' and Q'' are recursively of the same form. However, note that some queries may not be directly expressible using decision trees. For example, the query $(((A = 3) \wedge (B = 4)) \vee (B = 5))$, which may be the query with the smallest number of predicates cannot be directly expressed as a decision tree (since each node in a decision tree divides/splits the items into two mutually exclusive sets).

In our work we not only consider more general queries, but we also provide a taxonomy of different query families, showing how complexity of VDP relates to the query family. In addition, as mentioned earlier, we do not use experimental generality, but focus on metrics that do not need test data.

Query by Output: In [15], QBO is cast as a classification problem (into two classes: tuples in V, and tuples not in V) and decision trees are used to find a compact query. A decision tree is constructed in a greedy fashion by determining a "good" predicate to split the tuples into two classes. These two classes form the root nodes of two decision trees (each of which is constructed in a recursive fashion). The "goodness" of a predicate is assessed using a metric called Gini's index [14], and the best predicate is the one that reduces the Gini's index the most. Reference [15] does not explicitly try to optimize our metrics of succinctness or level or approximation, nor does it study the tradeoff between these metrics. Also, it does not consider the different families of queries that we do.

2. PROBLEM DEFINITIONS

In this paper, we assume that database D is a single relation R. We additionally assume that the relation R and the view V have the same schema. For V to be a meaningful view of R, it is necessary that $V \subseteq R$. We assume set semantics throughout the paper.

Since we deal with single table relations with both V and R having the same schema, any query Q can be represented as a selection condition. Any selection condition can be expressed as a DNF (Disjunctive Normal Form) formula, i.e., $Q \equiv c_1 \vee c_2 \vee \ldots \vee c_n$. Each c_i, called a *conjunctive clause*, is of the form $p_{i1} \wedge p_{i2} \wedge \ldots p_{im_i}$, where each p_{ij} is a *predicate*. We restrict our predicates to be equalities (e.g., $a = 5$), inequalities (e.g., $a \neq 5$) or ranges ($a \in [5, 10]$). If $Q \equiv c_1$, i.e., if Q contains a single conjunctive clause, then Q is a *conjunctive query* (CQ). A DNF selection condition is therefore equivalent to a *union of conjunctive queries* (UCQ).

For each family of queries that we consider, we are interested in obtaining Q such that: (1) $Q(R)$ is as *close* to V as possible (i.e., as *accurate* as possible): Intuitively, $Q(R)$ should miss out on few tuples from V, and should include few tuples not in V. (2) Q is as *succinct* as possible. Next we introduce a *difference* measure to quantify the closeness between V and $Q(R)$ (Section 2.1) and a metric of succinctness for queries (Section 2.2). Then, we formally define the specific problems addressed in this paper (Section 2.3), and the families of queries we consider (Section 2.4). Finally, in Section 2.5, we present a summary of our results. We use the following running example throughout the paper.

Movie	Director	Genre	Box office
Kill Bill	Quentin Tarantino	Action	Hit
Pulp Fiction	Quentin Tarantino	Comedy	Hit
Grindhouse	Quentin Tarantino	Horror	Flop
Jurassic Park	Steven Spielberg	Action	Hit
1941	Steven Spielberg	Comedy	Flop
The Village	M.N. Shyamalan	Horror	Flop
Sixth Sense	M.N. Shyamalan	Horror	Hit
The Godfather	F.F. Coppola	Action	Hit
Donnie Brasco	Mike Newell	Action	Hit
Snatch	Guy Ritchie	Comedy	Hit

Table 1: Relation Instance

Movie	Director	Genre	Box office
Kill Bill	Quentin Tarantino	Action	Hit
Pulp Fiction	Quentin Tarantino	Comedy	Hit
Sixth Sense	M.N. Shyamalan	Horror	Hit
Snatch	Guy Ritchie	Comedy	Hit

Table 2: View Instance

EXAMPLE 2.1. *Consider a relation R_{mov} giving information about movies, shown in Table 1. Also consider a relation V_{mov} in Table 2, obtained from R_{mov} through manual selection of tuples. Henceforth, we use R and V to abbreviate R_{mov} and V_{mov} respectively.*

2.1 Difference Metric

Given relations R, V, and a query Q, we say that $Q(R)$ is *close* to V if $Q(R)$ has few incorrect or missing tuples. The difference between two relations of the same schema (in this case $Q(R)$ and V) is captured by their symmetric set difference:

DEFINITION 2.2 (DIFFERENCE). *Given two relations R_1 and R_2, we say that the difference between R_1 and R_2 is given by $d(R_1, R_2) = |R_1 - R_2| + |R_2 - R_1|$.*

Note that the definition above can also be interpreted as the total number of tuples that need to be deleted or added to convert relation R_1 into relation R_2 (or vice versa). The smaller the difference $d(Q(R), V)$, the closer $Q(R)$ is to V.

There are situations when we may have an additional constraint that one of either $|V - Q(R)|$ or $|Q(R) - V|$ should be 0, i.e., we require $Q(R)$ to be either a superset or a subset of V. For instance, suppose the result of finding a view definition is given to a human for manual pruning. In such a case it may be critical to capture all result tuples in V, but extra tuples in $Q(R)$ can be removed manually. We consider difference metrics incorporating these additional constraints in Section 6.

2.2 Query Succinctness

We first introduce the notion of *size*, for conjunctive queries and unions of conjunctive queries, and then define succinctness in terms of size. We measure the size of a conjunctive query (CQ) as its number of predicates. We measure the size of a union of conjunctive queries (UCQ) as its number of unions. For UCQs, an alternative metric would include both number of unions and number of predicates in each clause. We studied this alternative metric as well for each of the problems we discuss in this paper, and the algorithms and complexity are similar. Additionally, if the average size of each conjunctive clause is similar, the number of unions is a good measure of the complexity of the queries. We mention differences whenever appropriate, but primarily use the metric based on number of unions. Thus we have:

Definition 2.3 (Size). *The size of a conjunctive query Q,* size(Q)*, is the number of predicates in Q. The size of a union of conjunctive queries Q,* size(Q)*, is the number of unions appearing in Q.*

Given two queries from the same family Q and Q', we say that Q is more succinct than Q' if Q has a smaller size.

Definition 2.4 (Relative-Succinctness). *Given two queries Q and Q' from a family $\mathcal{Q}$ of queries, we say that $Q \preceq Q'$, i.e., Q is at least as succinct as Q', if size(Q)≤size(Q'). We say that Q is more succinct than Q' if $Q \preceq Q'$ but not $Q' \preceq Q$.*

Example 2.5. *Consider a query Q for Example 2.1: (Director = Quentin Tarantino $\wedge$ Box Office = Hit) $\vee$ (Director = Guy Ritchie) $\vee$ (Movie = Sixth Sense). This query has three conjunctive clauses, i.e., two unions. Another query containing three unions is the following: (Movie = Snatch $\wedge$ Box Office = Hit) $\vee$ (Movie = Kill Bill) $\vee$ (Movie = Pulp Fiction) $\vee$ (Movie = Sixth Sense). Both these queries have $V = Q(R)$, but the first query has fewer unions, and is therefore more succinct.*

2.3 View Definition Problems

Next we enumerate the problems we study in the rest of the paper. The first problem considers whether given R and V, there is an exact view definition in a certain query family.

Definition 2.6 (Exact View Def. (EVD)). *Given relation R, view V, and a query family $\mathcal{Q}$, return a $Q \in \mathcal{Q}$ such that $V = Q(R)$, if such a Q exists.*

There may be several queries Q such that $V = Q(R)$, as we've seen in Example 2.5. Our next problem addresses the case when EVD doesn't have a unique solution, or when it has no solution at all. When EVD has multiple solutions, we want the most succinct one. When there is no exact view definition at all, as a first priority we want the *closest* query, and as a second priority we want the most succinct one.

Definition 2.7 (Best View Def. (BVD)). *Given relation R, view V, and a family of queries $\mathcal{Q}$, find and return any $Q \in \mathcal{Q}$ such that:*

(1) there is no $Q' \in \mathcal{Q}$ with $d(V, Q'(R)) < d(V, Q(R))$

(2) $\forall$ $Q' \in \mathcal{Q}$ with $d(V, Q'(R)) = d(V, Q(R))$, $Q \preceq Q'$.

In some cases, minimizing the difference may be less critical than obtaining a succinct view definition. For instance, an approximate view definition that gets close to V may suffice, but we would like the most succinct view definition that achieves the required approximation. The next problem finds the most succinct view definition that satisfies a minimum level of approximation specified by a threshold on the difference metric.

Definition 2.8 (Approximate View Def. (AVD)). *Given relation R, view V, approximation threshold τ, and a family of queries $\mathcal{Q}$, check if $\exists Q \in \mathcal{Q}$ such that:*

(1) $d(V, Q(R)) \leq \tau$

(2) if for any $Q' \in \mathcal{Q}$, $d(V, Q'(R)) \leq \tau$, then $Q \preceq Q'$.

and return any such Q if one exists.

For all three problems, we use the difference metric from Definition 2.2. Also recall from Section 2.1 the alternative difference metrics that require either $|V - Q(R)| = 0$ or $|Q(R) - V| = 0$, i.e., seeking view definitions that result in supersets or subsets of V respectively. In Section 6 we consider two variants of BVD: (1) **BVD-Sup**, equivalent to BVD but restricted to view definitions Q satisfying $Q(R) \supseteq V$; (2) **BVD-Sub**, equivalent to BVD but restricted to view definitions Q satisfying $Q(R) \subseteq V$.

Definition 2.9 (Approximation Ratio). *Given relation R, view V, a family of queries $\mathcal{Q}$ (and τ in the case of AVD), we say that a solution $Q \in \mathcal{Q}$ gives an α-approximation to BVD (AVD resp.) if Q satisfies (1) Condition-1 of Definition 2.7 (Definition 2.8 resp.); and (2) Condition-2 of Definition 2.7 (Definition 2.8 resp.) by replacing "$Q \preceq Q'$" with "size(Q)≤ α.size(Q')".*

2.4 Families of Queries

We consider the following families of queries:

- $\mathcal{CQ}^1_=$**:** A conjunctive query with a single *equality predicate* $A = a$.
- $\mathcal{CQ}^1$**:** A conjunctive query with a single *range predicate*, i.e., $A = a$ or $A \neq a$ or $A \in [a_1, a_2]$.
- $\mathcal{CQ}_=$**:** A conjunctive query with any number of equality predicates, but at most one predicate per attribute.
- $\mathcal{CQ}$**:** A conjunctive query with any number of range predicates, but at most one predicate per attribute.
- $\mathcal{UCQ}_=$**:** Unions of $\mathcal{CQ}_=$ queries.
- $\mathcal{UCQ}$**:** Unions of $\mathcal{CQ}$ queries.
- $\mathcal{UCQ}^k_=$**:** Union of at most k $\mathcal{CQ}_=$ queries ($k \geq 1$).
- $\mathcal{UCQ}^k$**:** Union of at most k $\mathcal{CQ}$ queries ($k \geq 1$).

Example 2.10. *(Director = M.N. Shyamalan) is a query in $\mathcal{CQ}^1_=$ (and other more general families), while (Director = M.N. Shyamalan $\wedge$ Genre = Horror) is a query in $\mathcal{CQ}_=$. (Movie = Snatch) $\vee$ (Movie = Kill Bill) $\vee$ (Movie = Pulp Fiction) $\vee$ (Movie = Sixth Sense) is a $\mathcal{UCQ}_=$ query, as well as a $\mathcal{UCQ}^4_=$ query.*

2.5 Summary of Results and Outline

Consider the EVD, BVD, and AVD problems for a relation R with M attributes and N tuples. Table 3 summarizes the results we have obtained for all the problems specified in Section 2.3, showing worst-case complexity and approximation ratios. (Some results from Table 3 are relatively straightforward, but still included in the paper for completeness.) For $\mathcal{UCQ}^k_=$ and $\mathcal{UCQ}^k$, since the size of the query is fixed (to k unions), we use an approximation metric different from Definition 2.9, requiring different techniques and analysis (discussed in Section 5).

Note that for some of the problems listed here, we analyze *data complexity*, i.e., the complexity on keeping schema size fixed and varying the size of the data (the number of

Family	EVD	BVD	AVD	BVD-Sup	BVD-Sub
$\mathcal{CQ}^1_=$	$O(MN)$	$O(MN)$	$O(MN)$	$O(MN)$	$O(MN)$
$\mathcal{CQ}^1$	$O(MN^2)$	$O(MN^2)$	$O(MN^2)$	$O(MN)$	$O(MN^2)$
$\mathcal{CQ}_=$	$O(MN)$	$O(MN^{M+1})$ NP-Hard(N,M) $O(MN^2)$ $\log N$-approx	$O(MN^{M+1})$ NP-complete(N,M) $O(MN^{\tau+2})$ $\log N$-approx	$O(MN.2^M)$	$O(M.N^{M+1})$
$\mathcal{CQ}$	$O(MN)$	$O(MN^{2M+1})$ NP-Hard(N,M) $O(MN^3)$ $\log N$-approx	$O(MN^{2M+1})$ NP-complete(N,M) $O(MN^{\tau+3})$ $\log N$-approx	$O(MN.2^M)$ NP-Hard(N,M)	$O(M.N^{2M+1})$ NP-Hard(N,M)
$\mathcal{UCQ}_=$	$O(MN)$	NP-complete(N,M) $PTIME(N^M)$ $\log N$-approx.	NP-complete(N,M)	N/A	N/A
$\mathcal{UCQ}$	$O(MN)$	NP-complete(N,M) $PTIME(N^{2M})$ $\log N$-approx.	NP-complete(N,M)	N/A	N/A
$\mathcal{UCQ}^k_=$	$O(MN^{Mk+2})$ NP-complete(N,M,k)	$O(MN^{Mk+2})$ NP-complete(N,M,k) PTIME $2k$-approx	N/A	$O(MN^{Mk+2})$ NP-complete(N,M,k)	$O(MN^{Mk+2})$ NP-complete(N,M,k)
$\mathcal{UCQ}^k$	$O(MN^{2Mk+2})$ NP-complete(N,M,k)	$O(MN^{2Mk+2})$ NP-complete(N,M,k) PTIME $2k$-approx	N/A	$O(MN^{2Mk+2})$ NP-complete(N,M,k)	$O(MN^{2Mk+2})$ NP-complete(N,M,k)

Table 3: Summary of Results.

tuples in R) as well as *combined complexity*, i.e., the complexity when neither schema size nor data size is fixed. We analyze these two cases separately because the data complexity may be different from the combined complexity for some problems, e.g., BVD for $\mathcal{CQ}$, where the data complexity is PTIME(N), while the combined complexity is NP-Complete (denoted as NP-Complete(N, M)). Similar notions were first introduced in [17].

Note that we don't separately consider the BVD-Sup and BVD-Sub problems for $\mathcal{UCQ}_=$ and $\mathcal{UCQ}$ (marked N/A in Table 3), as there is always a solution to EVD, and hence BVD's solution directly applies to BVD-Sup and BVD-Sub. Also note that the NP-Completeness results for $\mathcal{UCQ}^k_=$ and $\mathcal{UCQ}^k$ for EVD, BVD and BVD-Sup/Sub only hold if we allow the parameter k and the schema size M to vary along with N, the data size. If not, the problems are solvable in PTIME in N. Further, we don't consider AVD for $\mathcal{UCQ}^k_=$ and $\mathcal{UCQ}^k$ since the size of all these queries is fixed (k clauses), therefore, the solution to AVD is no different from the solution to BVD. The approximation result for AVD for $\mathcal{CQ}$ and $\mathcal{CQ}_=$ only holds for the special case when the view definition is required to give a superset of V, i.e., the analog of BVD-Sup for AVD.

In the remainder of the paper, we present details for all the results from Table 3, organized as follows. Section 3.1 considers the families of queries $\mathcal{CQ}^1_=$ and $\mathcal{CQ}^1$. Section 3.2 considers the families $\mathcal{CQ}_=$ and $\mathcal{CQ}$. Section 4 considers the families $\mathcal{UCQ}_=$ and $\mathcal{UCQ}$. Section 5 considers the families $\mathcal{UCQ}^k_=$ and $\mathcal{UCQ}^k$. The BVD-Sup and BVD-Sub problems are studied in Section 6.

3. FAMILIES OF CONJUNCTIVE QUERIES

We first consider the families of conjunctive queries with single predicates, i.e., $\mathcal{CQ}^1_=$ and $\mathcal{CQ}^1$ (Section 3.1). We then consider the families $\mathcal{CQ}_=$ and $\mathcal{CQ}$ of conjunctive queries with multiple predicates (Section 3.2).

3.1 Single Predicate

Since the results for single-predicate queries are simple in comparison to the other families of queries, we only state our main results, in the next two theorems. Details are presented in Appendix A.

THEOREM 3.1. *Given a relation instance R with M attributes and N tuples, and a view instance V, each of EVD, BVD, and AVD can be solved in $O(MN)$ for $\mathcal{CQ}^1_=$.*

THEOREM 3.2. *Given a relation instance R with M attributes and N tuples, and a view instance V, each of EVD, BVD, and AVD can be solved in $O(MN^2)$ for $\mathcal{CQ}^1$.*

Intuitively, for each of these problems, $d(V, Q(R))$ for all possible queries Q in the given family of queries needs to be computed. This computation is significantly aided by hashing, allowing all counts to be obtained in $O(MN)$.

3.2 Multiple Predicates

Next we consider the view definition problems for conjunctive queries, i.e., families $\mathcal{CQ}$ and $\mathcal{CQ}_=$. First, we present a connection between our problems and a set cover formulation, which is used in the rest of the section (Section 3.2.1). We then study EVD (Section 3.2.2), BVD (Section 3.2.3) and AVD (Section 3.2.4) problems respectively.

3.2.1 Set Cover Formulation

Given a relation R, and a view V, we create an instance of the set cover problem as follows: Construct the universal set $U = R - V$. Consider the set of single-attribute predicates $\mathcal{P}$ such that $P \in \mathcal{P}$ iff $P(R) \supseteq V$. For each $P \in \mathcal{P}$ we define a set $S_P = U - (P(R) \cap (R - V))$.

Intuitively, our aim is to cover all the elements of U, thereby eliminating all the incorrect tuples. We perform this cover by selecting a set of predicates $\mathcal{T} \subseteq \mathcal{P}$ such that $\cup_{P \in \mathcal{T}} S_P = U$. In other words, a cover of U gives a solution to the EVD for R and V: Since every element of U is covered, the corresponding set of predicates in conjunction does not select any tuple in $R - V$. Moreover, each predicate selects all tuples of V.

3.2.2 EVD

We first consider EVD for the family of queries $\mathcal{CQ}_=$. We will handle $\mathcal{CQ}$ subsequently.

Theorem 3.3. *Given a relation R and a view instance V, Algorithm 2 checks if there exists a solution to EVD for the query family $\mathcal{CQ}_=$ and finds one if there exists in $O(MN)$.*

The proof of this result may be found in Appendix Sec.B.1.

The result above can be easily extended for $\mathcal{CQ}$ with time complexity still $O(MN)$. For each attribute, we first pick the smallest superset range in the view as a predicate. We further check if there is a single value in the range present in the relation but missing in the view. (If so, we replace the range predicate with one predicate involving $\neq$ for the missing value. Note that if there are multiple values missing within the range, then there is no EVD.) This check can be done in $O(N)$ per attribute ($O(MN)$ overall) by using hashing to count the number of distinct values. We then check if the conjunctive clause thus formed is an EVD.

```
Require: $A_1, A_2, \ldots, A_n \leftarrow$ attributes of relation and view
Require: $b_1, b_2, \ldots, b_n \leftarrow$ sizes of view attribute domains
Require: $V \leftarrow$ view instance, $R \leftarrow$ relation instance
 1: $Q \leftarrow$ NULL
 2: for $i \in 1 \ldots n$ do
 3:     if $b_i == 1$ then
 4:         $v_i \leftarrow$ value of $A_i$ in view
 5:         $Q \leftarrow Q \wedge (A_i = v_i)$
 6:     end if
 7: end for
 8: if $|Q(R)| == |V|$ then
 9:     return $Q$
10: else
11:     return No EVD
12: end if
```

Algorithm 2: Algorithm to solve EVD for $\mathcal{CQ}_=$.

3.2.3 BVD

We start this section by a straightforward upper bound for BVD. We then present a lower bound, followed by an approximation algorithm.

Upper Bound

Let us consider the BVD problem for input R with N tuples and M attributes, and view V. To obtain an upper bound for BVD, we simply consider all possible conjunctive queries over R and pick the best possible one. R has $O(N^M)$ or $O(N^{2M})$ queries depending on whether our family of queries is $\mathcal{CQ}_=$ or $\mathcal{CQ}$. In $O(MN)$ we can compute $d(Q(R), V)$ for each query. If we assume the schema size M to be a fixed parameter and vary N, the size of the data, BVD's data complexity is PTIME, established by the following theorem. (We note that if R and V are *cross-products* of sets of values for attributes, BVD can be solved in $O(MN)$. Details of this special case appear in Appendix B.2.)

Theorem 3.4. *Given a relation R, view instance V, the BVD problem has a data complexity of $O(MN^{M+1})$ and $O(MN^{2M+1})$ for the family of queries $\mathcal{CQ}_=$ and $\mathcal{CQ}$ respectively.*

Example 3.5. *For Example 2.1, the BVD query for $\mathcal{CQ}_=$ is* (Director = Q.Tarantino $\wedge$ Box Office = Hit), *which has $d = 2$ due to two missing tuples.*

Lower Bound

We have seen an upper bound on the complexity of BVD. Our next result shows an exponential lower bound in terms of combined data and schema complexity of R. Recall that on fixing the number of attributes, the data complexity is polynomial in N. However, if we allow the number of attributes and the number of tuples in R to both vary, we show intractability. Our proof uses a relation R where the number of tuples and number of attributes are polynomials of each other.

Theorem 3.6. *Given R and V, the BVD problem is NP-Hard in N and M, for $\mathcal{CQ}_=$ and $\mathcal{CQ}$.*

Proof. We give a reverse construction based on the idea from Section 3.2.1. That is, for each instance of the set cover problem, we construct an instance of the BVD problem such that the solution of the BVD instance yields a solution to the set cover problem. Given an instance of the set cover problem consisting of the universal set $U = \{1, \ldots, n\}$, and subsets $S_1, \ldots, S_m$, we construct an input to BVD as follows (Note that the set cover problem is NP-hard even when the number of sets m is restricted to a polynomial of n):

$R(A_1, \ldots, A_m)$ contains a tuple t_i corresponding to each element $i \in U$. $t_i.A_j = 0$ if $i \notin S_j$. For every attribute value in R not assigned 0 based on the rule above, a distinct attribute value is assigned. (Note that there is no tuple marked with 0 in all attribute values, since we can assume without loss of generality that every element $i \in U$ appears in some subset S_j.) Add a special tuple t_0 to R where $\forall j, t_0.A_j = 0$. Let $V = \{t_0\}$.

Since V has a single tuple with all attribute values being 0, for BVD, we only need to consider queries with predicates of the form $A_j = 0$. Every such predicate corresponds to picking a set S_j for the set cover. Our goal reduces to finding the fewest such predicates such that each of the tuples $t_1, \ldots, t_n$ is not picked by at least one predicate, which is equivalent to finding the smallest cover for U.

The size of our constructed relation R is $O(mn)$, where the set cover problem has n elements and m attributes. Hence, BVD is NP-hard in combined data and schema size. $\square$

Approximation Algorithm

Now that we've established a lower bound on the complexity of BVD, we next study PTIME (in schema and data of R) approximation algorithms for solving the problem. We distinguish two cases: (1) When there is no solution to EVD; (2) When there exists a solution to EVD.

For Case (1), we employ the algorithm used for Theorem 3.4, i.e., looking at all possible queries and picking the best one. Our next result is an efficient approximation algorithm for Case (2). Given inputs R and V, we reduce BVD to the set cover problem based on the construction in Section 3.2.1. We then use Set Cover's greedy approximation algorithm [18]. Since the constructed set cover formulation has $O(N)$ elements, and $O(MN)$ sets (a predicate corresponding to each value of each attribute) for $\mathcal{CQ}_=$ and $O(MN^2)$ sets for $\mathcal{CQ}$, we have the following result.

Theorem 3.7. *Given relations R and V such that EVD has a solution for $\mathcal{CQ}_=$ ($\mathcal{CQ}$ resp.), there is an $O(MN^2)$-time ($O(MN^3)$-time resp.) $\log N$-factor approximation algorithm for BVD.*

3.2.4 AVD

We consider the AVD problem for $\mathcal{CQ}_=$ and $\mathcal{CQ}$. The worst-case upper bound result of BVD for these families of queries carries over for AVD as well. First we state the complexity of AVD, in the following theorem, and then present an approximation algorithm.

THEOREM 3.8. *Given R and V, the AVD problem is NP-complete in N and M, for $\mathcal{CQ}_=$ and $\mathcal{CQ}$.*

PROOF. NP-hardness of AVD follows from the reduction in the proof of Theorem 3.6, which also applies when $\tau = 0$. AVD is in NP because, given a query Q, in PTIME we can check whether $d(V, Q(R)) \leq \tau$, and then we check if any more succinct query Q' also has $d(V, Q'(R)) \leq \tau$. □

Next we present an approximation algorithm for AVD, under the *superset* special case described in Section 2. Suppose we are given a threshold τ along with R and V, and we want the most succinct query Q that guarantees $d(Q(R), V) \leq \tau$. We can approximately solve AVD using our set cover construction when we impose the condition that $Q(R) \supseteq V$, which we call the AVD-Sup problem: Given R and V, we construct an instance of the set cover problem as in Section 3.2.1. We are then interested in a solution that covers at least $\delta = (|R - V| - \tau)$ elements. Once again, we can use set cover's greedy approximation algorithm. However, we must solve the set cover for all possible choices of δ elements from the $|R - V|$ elements. We solve these $\binom{|R-V|}{\delta} \leq \binom{N}{\tau}$ set cover problems and pick the best, i.e., smallest, query among all of them.

THEOREM 3.9. *Given relations R and V, and threshold τ, there is an $O(MN^{\tau+2})$-time ($O(MN^{\tau+3})$-time resp.) $\log N$-factor approximation algorithm for the AVD-sup problem for $\mathcal{CQ}_=$ ($\mathcal{CQ}$ resp.).*

4. UNIONS OF CQ

We now study VDP for a union of conjunctive queries, i.e., query families $\mathcal{UCQ}$ and $\mathcal{UCQ}_=$.

4.1 EVD

The following straightforward theorem, proved in Appendix C, establishes the result for EVD.

THEOREM 4.1. *Given R, V, there exists a solution to EVD for $\mathcal{UCQ}$ and $\mathcal{UCQ}_=$ if and only if $V \subseteq R$.*

Therefore, EVD can be solved in $O(MN)$ time. Note that while the above theorem says that EVD always has a solution for $V \subseteq R$, the next example shows that the solution may not be unique.

EXAMPLE 4.2. *For Example 2.1, the queries (Movie = Snatch) $\vee$ (Movie = Kill Bill) $\vee$ (Movie = Pulp Fiction) $\vee$ (Movie = Sixth Sense) and (Movie = Snatch) $\vee$ (Movie = Kill Bill) $\vee$ (Director = Q. Tarantino $\wedge$ Box Office = Hit), are both EVDs for the view.*

4.2 BVD

Since there is a query Q' such that $Q'(R) = V$, the BVD query Q has to satisfy $Q(R) = V$ as well. Hence, BVD reduces to finding the most succinct Q such that $Q(R) = V$. In this section we focus on the family $\mathcal{UCQ}_=$, but the algorithms for $\mathcal{UCQ}$ are similar.

```
Require: $t_1, t_2, \ldots, t_v \leftarrow$ tuples in the view
 1: /* Generate */
 2: $S \leftarrow \emptyset$, $T \leftarrow \{t_1, t_2, \ldots, t_v\}$, $f \leftarrow false$
 3: while $f == false$ do
 4:     $S \leftarrow S \cup T$ (use the Exclusion Criterion)
 5:     $R \leftarrow \emptyset$
 6:     for $i \in T$ do
 7:         $R \leftarrow R \cup$ (sub-clauses of $i$ on dropping 1 predicate)
 8:     end for
 9:     discard conjunctive clauses containing incorrect tuples from
        $R$ (use the Pruning Criterion)
10:     $T \leftarrow R$
11:     $f \leftarrow (R == \emptyset)$
12: end while
13: /* Cover */
14: for all $Q$ such that $Q \subseteq S, |Q| = 1..v$ do
15:     if $d(Q(R), V) == 0$ then
16:         return $Q$
17:     end if
18: end for
```

Algorithm 3: Algorithm to solve the BVD.

4.2.1 Complexity

We now prove that finding a BVD is NP-Complete, by using a reduction from the Vertex Cover problem.

THEOREM 4.3. *Given R, V, finding a solution to BVD is NP-complete in the size of R's schema and data for $\mathcal{UCQ}_=$ and $\mathcal{UCQ}$.*

PROOF. We give a reduction from the NP-complete Vertex Cover problem [18]: Given an undirected graph $G(V, E)$, $V = \{v_1, \ldots, v_m\}$ and $E = \{e_1, \ldots, e_n\}$, determine the smallest set $V_{cov} \subseteq V$ such that all edges in E have at least one endpoint in V_{cov}. The vertex cover problem is NP-complete in N, when m and n are $poly(N)$. (Specifically, $m^2 \geq 2n \geq m$ is sufficient for NP-hardness.)

Given an instance of the vertex cover problem, we construct an instance of BVD as follows: Create a relation $R(A_1, \ldots, A_m)$ consisting of n tuples $t_1, \ldots, t_n$. $t_i.A_j = 0$ if and only if vertex v_j is an endpoint of edge e_i. All other entries in R are assigned distinct values. Further, let the view $V = R$. Now, to construct a solution to BVD, the only useful conjunctive clauses are of the form $(A_j = 0)$, which corresponds to selecting vertex v_j in the vertex cover. Any vertex cover can be represented as a union of conjunctive clauses corresponding to each vertex. Hence, there exists a vertex cover of size k if and only if there is a UCQ with k conjunctive clauses solving BVD. It is easy to see that the decision version of the problem is in NP, hence BVD is NP-complete in the size of the data and schema. □

Note that the same hardness proof above holds even if the metric for succinctness is not the number of unions, but is instead the total number of predicates in the DNF formula. The proof holds because only conjunctive clauses of size 1 are useful in the above reduction. As a result, both the metrics are equivalent (i.e., the total number of predicates = total number of clauses).

4.2.2 Two-phase Algorithm

Although BVD is NP-complete in general, Algorithm 3 describes a two-phase algorithm for an exhaustive search. Further, we can apply certain criteria to reduce the search space, as described shortly.[1] Our algorithm operates in two

[1]The worst-case guarantees don't change.

phases: *Generate* and *Cover*. The Generate phase generates all the potentical candidate conjunctive clauses in a top-down fashion (conjunctive clauses with m predicates, then $m-1$ predicates, and so on), starting with the tuples themselves as conjunctive clauses (line 2), and removing one predicate at a time to form smaller conjunctive clauses (line 7). (T is the set of conjunctive clauses generated in the previous iteration, S contains all the clauses generated so far, and R is the candidate set of clauses to be added in the next iteration.) These conjunctive clauses should not select any tuples not in the view (i.e., incorrect tuples) — this condition is checked in line 9. Instead, we could also use the following condition to directly prune good conjunctive clauses from bad ones. However, for this condition to be applied, we need to incur the cost of maintaining conjunctive clauses containing incorrect elements.

LEMMA 4.4 (PRUNING CRITERION). *If for a conjunctive clause with s predicates, there is a $(s+1)$ predicate conjunctive clause (with one extra predicate) which selects any tuples not found in V, then the conjunctive clause with s predicates can be removed from the set of candidate conjunctive clauses.*

Note that the pruning condition is remniscent of the pruning found in other domains, including finding all functional dependencies [6] and association rule mining [1].

The number of conjunctive clauses generated and added to S is governed by the size of the view, which may be much smaller than the relation. We could use the following criterion to further reduce the size of S.

LEMMA 4.5 (EXCLUSION CRITERION). *If a conjunctive clause with s predicates has been selected, then all conjunctive clauses with the same s predicates and $r, r \geq 1$ additional predicates can be removed from S.*

The Cover phase picks k conjunctive clauses (where k goes from $1..n$) from the candidate conjunctive clauses such that all tuples in V are covered/selected. All combinations of k conjunctive clauses from the set of candidate conjunctive clauses are tried for this purpose until one is found with $d = 0$ (line 15). The returned query is simply the union of those conjunctive clauses. Note that if we were to use the metric counting number of predicates in the DNF, then we would need to try all combinations of conjunctive clauses, and return the one with the smallest number of predicates.

Note that this phase is expensive computationally (even though we mentioned several methods to reduce the size of the candidate set S), so we describe an approximation algorithm to reduce the complexity in the next section.

Also note that the BVD algorithm is remniscent of the Quine-McCluskey procedure for finding minimal DNF for a boolean function [10] (and more generally other boolean formula minimization techniques [4]). The problem in that case is Σ_2^p-complete [16].

As an aside, note that we can use database properties to simplify the input to BVD, such as using the following rule. Our paper doesn't delve into exploring more such properties.

LEMMA 4.6 (FUNC. DEPENDENCY PRUNING). *Given R, V input to BVD/AVD, if R satisfies the FD $X \rightarrow Y$, then we can drop attributes $Y - X$ from R and V to get an equivalent formulation of the problem.*

4.2.3 Approximation Algorithm

We now describe an approximation algorithm for the Cover phase of the BVD algorithm.

THEOREM 4.7. *Given R with N tuples and M attributes, a view V, there exists a PTIME (in N) algorithm that finds a $\log N$-approximation to the optimal solution: If the optimal solution has k unions, our algorithm produces at most $k \log N$ unions.*

PROOF. In the following, we give an L-reduction from BVD to the set cover problem. We can then use the greedy algorithm for set cover that guarantees a $\log N$ factor approximation ratio [18].

There are a polynomial (in N) number of $\mathcal{CQ}_=$ queries over R: There are at most N possible predicates over each attribute, and hence at most $(N+1)^M$ conjunctive queries. Create an instance of set cover where the universe U is the set of all tuples in R. Each conjunctive query C forms a subset $S \subseteq U$ containing precisely the tuples selected by C. The optimal solution to the set cover problem gives the optimal UCQ containing the conjunctive queries selected in the set cover's solution.

At each stage the greedy algorithm picks the best conjunctive clause, i.e., one that covers maximum number of tuples from the view. The algorithm terminates when all the tuples in the view have been selected. $\square$

Note that for the metric that counts the total number of predicates in the DNF formula, we can describe a similar approximation algorithm using a reduction to the *weighted set cover* problem [18], where the cost of each set corresponding to a conjunctive clause is the number of predicates present in the conjunctive clause. In this case at each stage the greedy algorithm picks the conjunctive clause that has the largest ratio of the number of new tuples from the view covered to the cost of the clause.

4.3 AVD

We first present the complexity of finding an AVD, then explain how we can adapt Algorithm 3 for AVD.

4.3.1 Complexity

COROLLARY 4.8. *Finding a solution to AVD for $\mathcal{UCQ}_=$ and $\mathcal{UCQ}$ is NP-complete in the schema size and the data size.*

The hardness proof is easy to see on setting $\tau = 0$ for the reduction in Theorem 4.3. Additionally, AVD is in NP because given a solution Q to the AVD which has k unions, we can test all possible UCQs Q' of size upto k (a polynomial in n) and check if $d(Q(R), V) < d(Q'(R), V)$, all in PTIME.

4.3.2 Two Phase Algorithm

We now describe a modification of Algorithm 3 for AVD. As before, the algorithm has two phases, Generate and Cover. In the Generate phase, we can apply a natural generalization of the pruning criterion. In addition, we can apply the following criteria, which are different from the criteria used for BVD.

LEMMA 4.9 (INCLUSION CRITERION). *If a conjunctive clause Q selects $> \tau$ tuples not present in the view, then Q is not included in S.*

LEMMA 4.10 (EXCLUSION CRITERION). *If a conjunctive clause Q with s predicates has been selected, then all conjunctive clauses with Q' the same s predicates and $r, r \geq 1$ additional predicates that have $d(Q'(R), V) = d(Q(R), V)$ can be removed from S.*

5. BOUNDED UNIONS

In this section we consider the families of queries where the number of unions is bounded by k, i.e., the families $\mathcal{UCQ}_=^k$ and $\mathcal{UCQ}^k$. Section 5.1 considers the case when k is not fixed and Section 5.2 considers the case when k is a fixed parameter. Recall, since the AVD problem reduces to the BVD problem when the query size is fixed, we do not separately consider the AVD problem.

5.1 Variable Parameter

The following theorem establishes the complexity of solving EVD and BVD when k is a variable parameter.

THEOREM 5.1. *Given input relation R with N tuples and M attributes, and a view instance V, the EVD and BVD problems are NP-complete in N, M and k for $\mathcal{UCQ}_=^k$ and $\mathcal{UCQ}^k$.*

PROOF. The problem is clearly in NP: Given a $\mathcal{UCQ}_=^k$ or $\mathcal{UCQ}^k$ query Q, we can check in PTIME whether Q is a solution to EVD or if there are any queries with a smaller difference. The hardness result follows from the hardness of BVD for $\mathcal{UCQ}_=$ and $\mathcal{UCQ}$ (Theorem 4.3). In the proof of Theorem 4.3, we constructed an input instance R and V where checking if there was a BVD $\mathcal{UCQ}$ (or $\mathcal{UCQ}_=$) with at most k unions was NP-hard. This instance can be reduced to checking if there is an EVD for $\mathcal{UCQ}^k$ or if there is a BVD for $\mathcal{UCQ}^k$ with difference 0. $\square$

5.2 Fixed Parameter

If k is fixed, we have the following theorem.

THEOREM 5.2. *Given a relation R and a view V, the BVD and EVD problem can be solved in $O(MN^{Mk+2})$ for $\mathcal{UCQ}_=^k$ and in $O(MN^{2Mk+2})$ for $\mathcal{UCQ}^k$.*

The theorem follows from the fact that all candidate queries in $\mathcal{UCQ}_=^k$ (in $\mathcal{UCQ}^k$ resp.) can be enumerated in $\binom{N^M}{k}$ ($\binom{N^{2M}}{k}$ resp.) and it takes at most $O(MN^2)$ to calculate the difference for each of these queries.

Since the approach above is exponential in M and k, we look for an approximate solution that is more tractable for the BVD problem. There are two main results in this section: a greedy approximation algorithm for BVD (Section 5.2.1), and an inapproximability result on a family of greedy algorithms (Section 5.2.2). As explained next, our greedy algorithm gives an approximation ratio under a weaker notion than approximating the difference metric. However, the inapproximability result from Section 5.2.2 shows that no greedy algorithm can achieve a good approximation ratio under the stronger notion.

Given input R, V, our result shows that the greedy algorithm gives a query Q_{algo} with a good approximation ratio for one of two metrics: (i) the difference metric between Q_{algo} and V, and (ii) the good-bad metric, i.e., the total number of tuples in V produced by Q_{algo} minus the number of tuples not in V produced by Q_{algo}. Define:

$$d_{algo} = min\{|V|, |V - Q_{algo}(R)| + |Q_{algo}(R) - V|\}$$

$$gb_{algo} = |V| - d_{algo}$$

The metrics for the optimal solution, d_{opt} and gb_{opt}, are defined similarly. Intuitively, d measures the difference, while gb measures the total number of "good tuples" (tuples in V) returned by the query minus the number of "bad tuples" (tuples not in V) returned by the query. Let the approximation ratios based on d and gb be r_d and r_{gb} respectively. That is,

$$r_d = \frac{d_{algo}}{d_{opt}} \qquad and \qquad r_{gb} = \frac{gb_{opt}}{gb_{algo}}$$

Ideally, for any algorithm, we want r_d and r_{gb} to be as small as possible, but Section 5.2.2 shows that greedy algorithms cannot guarantee good approximations under any of these metrics independently. Next, in Section 5.2.1 we consider approximating $\alpha = min\{r_d, r_{gb}\}$.

5.2.1 Approximation Guarantee

Given input R and V, we employ the following greedy algorithm, GREEDYKUCQ, to solve BVD for $\mathcal{UCQ}_=^k$ and $\mathcal{UCQ}^k$: GREEDYKUCQ iteratively picks the *best* conjunctive query until k conjunctive queries are picked. At each step, the best conjunctive query is the one that maximizes the gain in the gb metric, i.e., reduces the difference metric by the most. That is, we pick the conjunctive query that gives the largest number of new good tuples (tuples in V) minus the number of new bad tuples (tuples not in V). The algorithm terminates when k conjunctive clauses have been picked or when there is no query with gain > 0.

When R has N tuples and M attributes, GREEDYKUCQ runs in polynomial-time in N and k: There are $O(N^M)$ conjunctive queries in $CQ_=$ ($O(N^{2M})$ for CQ resp.), and at each iteration it takes time $\sim O(N^M)$ ($O(N^{2M})$ resp.) to pick the next best conjunctive clause, and we have at most k iterations. The following theorem establishes an approximation guarantee for GREEDYKUCQ.

THEOREM 5.3. *Given input R, V,* GREEDYKUCQ *gives a solution to the BVD problem satisfying $\alpha = min\{r_d, r_{gb}\} \leq (2k-1)$, for $\mathcal{UCQ}_=^k$ and $\mathcal{UCQ}^k$.*

PROOF. Suppose $V_{opt} = Q_{opt}(R)$, where Q_{opt} is the optimal solution to the BVD. Let $y = |V \cap V_{opt}|$, $x = |V_{opt} - V|$, and $z = |V - V_{opt}|$. Therefore, we have:

$$d_{opt} = (x + z) \tag{1}$$

$$gb_{opt} = (y - x) \tag{2}$$

Since k conjunctive queries put together give y good tuples (i.e., tuples in V, and x bad tuples (i.e., tuples not in V), there must exist a conjunctive query Q^* that gives at least $\frac{y}{k}$ good tuples and at most x bad tuples. We show that our result holds for the query Q^*, and since Q_{algo} is at least as good as Q^*, our result follows.

$$d_* \leq (z + y - \frac{y}{k}) + x = (z + x + y(1 - \frac{1}{k})) \tag{3}$$

$$gb_* \geq (\frac{y}{k} - x) \tag{4}$$

Using Equations 1 and 3, we have:

$$r_d \leq \frac{(z + x + y(1 - \frac{1}{k}))}{x + z} \tag{5}$$

Using Equation 2 and 4, we have:

$$r_{gb} \leq \frac{y - x}{\frac{y}{k} - x} \tag{6}$$

Suppose for some parameter β:

$$\frac{(y - x)}{\frac{y}{k} - x} = \beta \tag{7}$$

Then, we have

$$(\frac{y}{k} - x) = \frac{y - x}{\beta} \Rightarrow \frac{y}{x} = \frac{k(\beta - 1)}{\beta - k} \tag{8}$$

Substituting the above in Equation 5, we get:

$$\begin{aligned} r_d \leq & \quad \frac{z+x+x(1-\frac{1}{k})(\frac{k(\beta-1)}{\beta-k})}{z+x} \\ = & \quad \frac{z+x+\frac{x(k-1)(\beta-1)}{\beta-k}}{z+x} \leq \quad (1 + \frac{(k-1)(\beta-1)}{\beta-k}) \end{aligned} \tag{9}$$

Therefore, from Equations 6, 7, and 9, we have:

$$(r_{gb} \leq \beta) \ and \ (r_d \leq (1 + \frac{(k-1)(\beta-1)}{\beta-k})) \tag{10}$$

Therefore,

$$\alpha \leq min\{\beta, (1 + \frac{(k-1)(\beta-1)}{\beta-k})\} \tag{11}$$

The minimum of the two expressions above is always $\leq 2k$. This follows from the fact that the second expression is monotonically decreasing for $\beta \geq k$, and we get $\beta = (2k-1)$ by solving the following:

$$\beta = (1 + \frac{(k-1)(\beta-1)}{\beta-k})$$

□

5.2.2 Lower Bound

The following theorem states the straightforward result that GreedyKUCQ gives the optimal solution to BVD for $k = 1$ for both the d metric as well as the gb metric.

Theorem 5.4. *Given inputs R, V, GreedyKUCQ gives the optimal solution to the BVD problem for $\mathcal{UCQ}^1_=$ and $\mathcal{UCQ}^1$, under the difference metric d as well as the "good-bad" metric gb.*

Our next result, proved in Appendix D, shows that for $k > 1$, GreedyKUCQ can independently perform poorly for the gb and the d metrics, thus justifying the approximation metric of α for GreedyKUCQ.

Theorem 5.5. GreedyKUCQ *can give an arbitrarily bad approximation ratio when the gb and d metrics are considered independently.*

- *There exists an input R_{gb} and view instance V_{gb} with N tuples, such that for the BVD problem for $\mathcal{UCQ}^k_=$ and $\mathcal{UCQ}^k$, for $k > 1$, using* GreedyKUCQ*, we have $gb_{opt} \sim N$, $gb_{greedy} = 0$, and therefore get an approximation ratio $r_{gb} \to \infty$.*
- *There exists an input R_d and view instance V_d with N tuples, such that for the BVD problem for $\mathcal{UCQ}^k_=$ and $\mathcal{UCQ}^k$, for $k > 1$, using* GreedyKUCQ*, we get an approximation ratio $r_d = N(1 - \frac{1}{k})$.*

We now prove that we can perform arbitrarily bad if we use any greedy algorithm with a greedy function from a large class of functions. Consider a family of algorithms GreedyLin with linear greedy functions : $f(g, b) = \alpha g - \beta b$, where g is the number of tuples in V returned by a conjunctive query Q, b is the number of tuples not in V returned by Q, and α, β are positive constants. Also, we assume that at each step the function only evaluates conjunctive clauses based on the "new" tuples, i.e., the ones that have not been selected by the conjunctive clauses picked already.

Additionally, the algorithm could stop (before k unions are picked) if the conjunctive clause that is selected by the algorithm has no gain (i.e., number of new good tuples - number of new bad tuples < 0), could be constrained to always return k unions or could stop when there are no tuples left in V that have not been covered by the conjunctive clauses picked. (Our next result applies to all the aforementioned stopping conditions.) Also, note that GreedyKUCQ $\in$ GreedyLin.

Theorem 5.6. *Given inputs R with N tuples and M attributes, V, for the BVD problem for $\mathcal{UCQ}^k_=$ and $\mathcal{UCQ}^k$, for $k > 1$, for any* GreedyLin *algorithm G using a greedy function $f(g, b) = \alpha g - \beta b$, (1) There is a case for which $r_{gb} \to \infty$. (2) There is a case for which $r_d = O(N)$, i.e, $r_d \to \infty$ when $N \to \infty$.*

Proof.
(1) Note that $f(\gamma, \gamma) > f(\gamma - 1, \gamma)$ holds for any γ. Consider the following situation. We have two sets of conjunctive clauses, Set I: k conjunctive clauses each of which covers $\frac{N}{k}$ distinct good tuples and $\frac{N}{k}$ distinct bad tuples, and Set II: k conjunctive clauses each of which covers $\frac{N}{k}$ distinct good tuples and the same $\frac{N}{k} + 1$ bad tuples. Any GreedyLin algorithm would pick only conjunctive clauses from Set I. (It may choose to not pick any conjunctive clauses, or stop once k conjunctive clauses have been picked.) In any case, we have $gb_{greedy} = 0$. However, the optimal algorithm would pick k unions from Set II, giving rise to $gb_{opt} = N(1 - \frac{1}{k}) - 1$. Thus, for $k > 1$, $r_{gb} = \infty$.
(2) Consider the following situation. We have two sets of conjunctive clauses, Set I: a single conjunctive clause which covers all N good tuples and $\frac{\alpha}{\beta}(N - \frac{N}{k})$ bad tuples, and Set II: k conjunctive clauses each of which covers $\frac{N}{k}$ distinct good tuples and the same 1 bad tuple. Any GreedyLin algorithm would pick only conjunctive clauses from Set I, for the following reason:

$$\alpha \cdot N - \beta \cdot \frac{\alpha}{\beta}(N - \frac{N}{k}) > \alpha \frac{N}{k} - \beta$$

If the GreedyLin algorithm picks no clause, $d_{greedy} = N$. If it picks just one clause (from Set I), $d_{greedy} = N - (N - \frac{\alpha}{\beta}(N - \frac{N}{k})) = \frac{\alpha}{\beta}N(1 - \frac{1}{k})$. On picking additional conjunctive clauses, d_{greedy} is strictly greater, since the same good tuples would be covered, but additional bad tuples get covered. Thus in any case, we have $d_{greedy} = O(N)$. However, the optimal algorithm would pick k unions from Set II, giving rise to $d_{opt} = N - k\frac{N}{k} + 1 = 1$. Thus, $r_d = O(N)$. □

Note that these results would also hold for a much more general class of functions. In particular, any greedy algorithm that uses a function that satisfies $f(\gamma, \gamma) > f(\gamma - 1, \gamma)$ or $f(\gamma, \gamma) > f(\gamma, \gamma + 1)$ for any $\gamma = \alpha N < \frac{N}{k}$ would have $gb_{opt} = O(N)$ and $r_{gb} = \infty$.

Note that the BVD problem for $\mathcal{UCQ}^k_=$ is reminiscent of the red-blue cover problem [5] and the partial set cover problem [11]. Both these problems define a collection of elements with some positive elements and some negative elements, and a collection of sets each of which cover some positive and some negative elements. Sets need to be chosen such that the positive elements are covered without too many negative elements. However, both of these problems do not have a constraint on the number of sets that can be picked, and therefore represent slightly different problems involving different techniques and approximations.

6. BEST SUBSET AND SUPERSET VD

In this section, we consider variants of the BVD problem with certain additional constraints, for the different families of queries. The first problem we consider is that of the best query that results in a subset of the given view instance. This case is especially important if we cannot afford to have any extraneous tuples in our query result as compared to the view instance (i.e., no false positives), but we are allowed to miss a few tuples.

Definition 6.1 (best subset view def. (bvd-sub)). *Given relation R, view V, and a family of queries $\mathcal{Q}$, find $Q \in \mathcal{Q}$ such that all three of the following hold*

1. *$Q(R) \subseteq V$*
2. *there is no $Q' \in \mathcal{Q}, Q'(R) \subseteq V$ with $d(V, Q'(R)) < d(V, Q(R))$*
3. *for all $Q' \in \mathcal{Q}, Q'(R) \subseteq V$ with $d(V, Q'(R)) = d(V, Q(R))$, $Q \preceq Q'$.*

The second problem we consider is that of the best query that is a superset of the given view instance. This case is especially important if we cannot afford to have miss any tuples from the view instance (i.e., no false negatives).

Definition 6.2 (best superset view def. (bvd-sup)). *Given relation R, view V, and a family of queries $\mathcal{Q}$, find $Q \in \mathcal{Q}$ such that*

1. *$V \subseteq Q(R)$*
2. *there is no $Q' \in \mathcal{Q}, V \subseteq Q'(R)$ with $d(V, Q'(R)) < d(V, Q(R))$*
3. *for all $Q' \in \mathcal{Q}, V \subseteq Q'(R)$ with $d(V, Q'(R)) = d(V, Q(R))$, $Q \preceq Q'$.*

We state the following theorem without proof:

Theorem 6.3. *If a relation instance R and a view instance V have an EVD for a family of queries $\mathcal{Q}$ (i.e., $\exists Q \in \mathcal{Q}$ such that $Q(R) = V$), then the solutions of the problems BVD, BVD-Sup and BVD-Sub are the same.*

Based on the above theorem, we don't need to separately study BVD-Sup and BVD-Sub for $\mathcal{UCQ}_=$ and $\mathcal{UCQ}$. Also note that the same approach used for EVD and BVD in Theorem 5.1 may be used for solving BVD-Sup and BVD-Sub for $\mathcal{UCQ}^k$ and $\mathcal{UCQ}^k_=$. Next we describe algorithms to solve the BVD-Sup and BVD-Sup problems for each conjunctive query family.

6.1 CQs with a single predicate

BVD-Sup Problem: We first consider the BVD-Sup problem. For conjunctive queries with only equalities, we simply consider the attributes that are constants in the view and form CQs out of them. We have the following theorem.

Theorem 6.4. *Given a relation instance R and a view instance V, the procedure above finds the BVD-Sup over queries in $\mathcal{CQ}^1_=$ in $O(MN)$.*

Full details of how clauses corresponding to each value of each attribute are evaluated and the proof of the theorem above are in Appendix E.1. Instead, we give details for queries in $\mathcal{CQ}^1$, which subsume the results for $\mathcal{CQ}^1_=$.

For $\mathcal{CQ}^1_=$, we consider the smallest superset ranges of attribute values in the view V. If we were to include ranges, then for any attribute A_i, we could have the conjunctive clause $Q = j \leq A_i \leq k$. However, we restrict $j = \min_{A_i \in V}\{A_i\}$ and $k = \max_{A_i \in V}\{A_i\}$, because any range that does not include all of $[j, k]$ would not satisfy the condition of $Q(R) \subseteq V$, and any range that contains other elements cannot be better because it can only include tuples that are not in V. Let the proportion of tuples with values in $[j, k]$ for attribute A_i be p_i in the relation R. Thus, we would have $d(V, Q(R)) =$

$$d_i = p_i * |R| - |V|$$

The conjunctive clause that minimizes d_i above over all $1 \leq i \leq n$ is the BVD-Sup if we include range queries.

The procedure above makes one pass of each column to compute the min and the max ($O(MN)$), plus one pass per attribute to compute p_i ($O(MN)$), thus the complexity of the procedure is $O(MN)$.

Extending the procedure for $\neq$ is also $O(MN)$: For each attribute, count the number of values of the attribute in R that hash to the same location as a value for the same attribute in V. We can count this number in $O(MN)$. The difference between this number and the size of V is the difference when we have an $\neq$ predicate with that attribute.

Theorem 6.5. *Given a relation instance R and a view instance V, the procedure above finds the BVD-Sup over queries in $\mathcal{CQ}^1$ in $O(MN)$.*

BVD-Sub Problem: All queries Q in $\mathcal{CQ}_=$ for which the number of tuples in V that are selected by Q is the same as the number of tuples in R that are selected by Q are potential candidates for the BVD-Sub problem. We have the following theorem.

Theorem 6.6. *Given a relation instance R and a view instance V, the procedure above finds the BVD-Sub over queries in $\mathcal{CQ}^1_=$ in $O(MN)$.*

The details of the potential candidate solutions for the BVD-Sub problem and the proof of the theorem are in Appendix F.

For ranges, we have a similar check of cardinalities to shortlist potential queries: Queries corresponding to ranges for any given attribute are considered for which the number of tuples in the relation R with the attribute having a value in that range is the same as the number of tuples in the view V with the attribute having a value in the same range.

For ranges, for any attribute A_i, we could have the conjunctive clause $Q = j \leq A_i \leq k$. Let the proportion of tuples with values in $[j,k]$ for attribute A_i be $p_i^{j,k}$ in the relation R and $q_i^{j,k}$ in the view V. Note that the number of tuples in $[j,k]$ in V are the same as the number of tuples in R, and thus $p_i^{j,k} * |R| = q_i^{j,k} * |V|$. Thus, we would have $d(V, Q(R)) =$

$$d_i^{j,k} = (1 - q_i^{j,k}) * |V|$$

Let the attribute $A_{\bar{i}}$ and values $\bar{j}, \bar{k}$ be the ones that minimizes $d_i^{j,k}$ over all $1 \leq i \leq n, 1 \leq j \leq k \leq m_i$. The solution of the BVD-Sub problem is the range $Q = \bar{j} \leq A_{\bar{i}} \leq \bar{k}$.

The procedure above makes 1 pass of each column to compute $q_i^{j,k} = \Sigma_{l=1}^{k} q_i^l - \Sigma_{l=1}^{j-1} q_i^l$ and $p_i^{j,k}$ for all attribute value pairs. The statistics can be collected in $O(MN)$ using hashing, and in $O(MN^2)$ we can compute $d_i^{j,k}$ for all i, j, k and also make sure that the number of tuples in V and R for that attribute value pair is the same. Thus, we have a complexity of $O(MN^2)$. Note that the complexity of adding $\neq$ predicates is $O(MN)$ as well.

Theorem 6.7. *Given a relation instance R and a view instance V, the procedure above finds the BVD-Sub over queries in $\mathcal{CQ}_=^1$ in $O(MN^2)$.*

Note that we can speed up the procedure by noticing that if the number of tuples in range $[j,k]$ are the same in R and V, then we do not need to consider any sub-range $[j+\delta_1, k-\delta_2]$.

Example 6.8. *The BVD-Sup query in $\mathcal{CQ}_=^1$ for our movies example is (Box Office = Hit), while the BVD-Sub query is one of many queries, including (Movie = Snatch) or (Director = Guy Ritchie). Note that the BVD-Sup in this case has $d = 3$, while the BVD-Sub has $d = 3$ as well.*

6.2 CQs with Multiple Predicates

The NP-hardness of BVD-Sup and BVD-Sub follows from the result of Theorem 3.6, since our reduction to BVD had an EVD. Next we consider upper bounds for each of these problems.

BVD-Sup: If we assume that size is the same for all conjunctive queries, we can find the BVD-Sup over queries in $\mathcal{CQ}$ and $\mathcal{CQ}_=$ in $O(MN)$. We omit the details here — they may be found in Appendix E.2.

Example 6.9. *The BVD-Sup query in $\mathcal{CQ}_=$ for the Movies example is not the same as the BVD (and therefore has a larger difference from the view V), but is in fact, the same as the BVD-Sup query in $\mathcal{CQ}_=^1$: (Box Office = Hit).*

To find the BVD-Sup for our original definition of size (based on number of attributes in the query), we first construct a conjunctive query formed by selecting all predicates corresponding to attributes that are constants in the view. We then enumerate all its subsets of predicates, and select the best one. In practice, we can perform a "smarter" top-down search eliminating predicates from the query one at a time and terminating the search when any subset of attributes is not a solution to BVD-Sup. The top-down search, however, does not improve the worst-case complexity.

Theorem 6.10. *The worst case complexity for BVD-Sup is $O(MN.2^M)$ for $\mathcal{CQ}_=$ and $\mathcal{CQ}$.*

BVD-Sub: For finding the BVD-Sub query in the family $\mathcal{CQ}_=$ and $\mathcal{CQ}$, we enumerate all candidate queries that are subsets of the view V and pick the one with the smallest difference. Note also that if we are evaluating queries in a top down fashion, we can use the following lemmas.

Lemma 6.11 (Stopping Criterion). *If there is no query Q with k predicates that satisfies $Q(R) \subseteq V$, then there are no queries Q with $< k$ that satisfies $Q(R) \subseteq V$.*

Lemma 6.12 (Pruning Criterion). *The only queries Q with k predicates that need to be considered are those that are formed from those in $k+1$ by eliminating one predicate.*

We make one pass in $O(NM)$ to compute the difference for each conjunctive clause (there are atmost $O(N^M)$ conjunctive clauses in $\mathcal{CQ}_=$ and atmost $O(N^{2M})$ conjunctive clauses in $\mathcal{CQ}$), thus we have:

Theorem 6.13. *The complexity of BVD-Sub on $\mathcal{CQ}_=$ is $O(MN^{M+1})$ and on $\mathcal{CQ}$ is $O(MN^{2M+1})$.*

Example 6.14. *The BVD-Sub query in $\mathcal{CQ}_=$ for the Movies example is the same as the BVD, i.e., (Director = Tarantino $\wedge$ Box Office = Hit).*

7. CONCLUSIONS AND FUTURE WORK

This paper addressed the view definitions problem (VDP), considering succinctness and approximation constraints and a variety of query families. We identified three subproblems, EVD, BVD and AVD, and studied the complexity of solving each of the three subproblems for each query family. We provided polynomial-time optimal algorithms for the tractable cases and approximation algorithms for the intractable cases. Table 3 in Section 2.5 summarizes our findings for all the problems.

Some specific open problems remain, such as an approximation algorithm for AVD for unions of conjunctive queries. More generally, in this paper we considered only views derived by selection predicates over a single relation. Extending our results to arbitrary select-project-join (or beyond) queries is an important avenue of future work.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. pages 487–499, 1994.
[2] Philip A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.
[3] P. Bohannon, E. Elnahrawy, W. Fan, and M. Flaster. Putting context into schema matching. In *VLDB '06*.
[4] Robert King Brayton, Alberto L. Sangiovanni-Vincentelli, Curtis T. McMullen, and Gary D. Hachtel. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.
[5] R. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *SODA '00*, pages 345–353.
[6] H. G-Molina, J. Widom, and J. D. Ullman. *Database Systems: The Complete Book*. Prentice-Hall, 2002.

[7] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin*, 18(2), 1995.

[8] L. Haas. The theory and practice of information integration. In *Proc. of ICDT*, 2007.

[9] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 4, 2000.

[10] E. J. McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 1956.

[11] P. Miettinen. On the positive–negative partial set cover problem. *Inf. Process. Lett.*, 108(4):219–221, 2008.

[12] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[13] Pierre Senellart and Georg Gottlob. On the complexity of deriving schema mappings from database instances. In *Proc. PODS*, pages 23–32, Vancouver, Canada, June 2008.

[14] P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Addison-Wesley, 2006.

[15] Q. T. Tran, C. Y. Chan, and S. Parthasarathy. Query by output. In *Proc. of ACM SIGMOD*, 2009.

[16] C. Umans, T. Villa, and A. L. Sangiovanni-Vincentelli. Complexity of two-level logic minimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(7):1230–1246, 2006.

[17] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC '82*.

[18] V. V. Vazirani. *Approximation Algorithms.* Springer-Verlag, 2001.

APPENDIX

A. SINGLE EQUALITY PREDICATE

Consider the problem of finding a query with a selection condition consisting of a single conjunctive clause containing one predicate, i.e., query families $\mathcal{CQ}^1_=$ and $\mathcal{CQ}^1$. In this case all queries are of the same size, and therefore are all equally desirable. Therefore, we try to find the query Q in each of these query families that minimizes $d(Q(R), V)$. This query would then be the BVD for this family. If in fact the BVD query Q is such that $d(Q(R), V) = 0$, then Q is also an EVD. Additionally, if $d(Q(R), V) \leq \tau$, then Q is also an AVD — Note that the second condition in the AVD problem definition is trivially true because $Q_1 \preceq Q_2$ is true for all queries (since all queries are equivalent). Thus, we now proceed to give algorithms to find the BVD for each subfamily of the family of Single Predicate Conjunctive Queries.

We now introduce some notation. Let the attributes of R be $A_1, A_2, \ldots, A_m$. Without loss of generality, for any attribute A_i, let the domain $\mathcal{A}_i$ be $\{1, 2, \ldots m_i\}$.

Single Equality-Predicate CQ

We first consider the BVD problem for $\mathcal{CQ}^1_=$. Let the proportion of tuples with value j for attribute A_i be p_i^j in the relation R and q_i^j in the view V. Then, if we were to pick $(A_i = j)$ as the selection condition for query Q, then $d(V, Q(R))$ would be:

$$d_i^j = p_i^j * |R| - q_i^j * |V| + (1 - q_i^j) * |V|$$

We simply pick the attribute A_i and value j that minimizes d_i^j above. Let this value d_i^j be d_{min}.

While collecting statistics for each value in $\{1, 2, \ldots, m_i\}$ corresponding to attribute A_i, we scan the ith column of both R and V. If we assume hashing is an $O(1)$ operation, we get statistics per column in $O(N)$, where N is the number of tuples. There are M such columns. Computing values d_i^j for all i, j is another $O(MN)$ operation. Thus the complexity is $O(MN)$. On the other hand, if we do not assume hashing is an $O(1)$ operation, we can sort the column in $O(N \log N)$, giving a complexity of $O(MN \log N)$.

THEOREM A.1. *Given a relation instance R and a view instance V, the procedure above finds the EVD/BVD/AVD over queries in $\mathcal{CQ}^1_=$ in $O(MN)$.*

This entry is listed in Table 3 in the first 3 columns of the first row.

For Example 2.1, the best query (with $d = 3$) is any of (Genre = Comedy) – two missing tuples, one extra tuple, (Director = Q. Tarantino) – two missing tuples, one extra tuple, or (Box Office = Hit) – three extra tuples.

Single Predicate CQ

We now consider the BVD problem for $\mathcal{CQ}^1$, by picking the best query from queries of the form A_i *op* j, for each *op* in $\{=, \neq\}$. We then also consider ranges, for predicates of the form $A_i \in [j, k]$.

Consider inequality predicates, i.e., for any attribute A_i, we could have the query $Q \equiv (A_i \neq j)$. Thus $d(V, Q(R)) =$

$$d_i^{\neq j} = (1 - p_i^j) * |R| - (1 - q_i^j) * |V| + (q_i^j) * |V|$$

Let the attribute A_i and value j be the one that minimizes $d_i^{\neq j}$ above over all $1 \leq i \leq n, 1 \leq j \leq m_i$. Let this value $d_i^{\neq j}$ be $d^{\neq}_{min}$. The complexity of computing this value is the same as the complexity for equality predicates.

Consider range predicates, i.e., for any attribute A_i, we could have the query $Q = j \leq A_i \leq k$. Let the proportion of tuples with values in $[j, k]$ for attribute A_i be $p_i^{j,k}$ in the relation R and $q_i^{j,k}$ in the view V. Thus, we would have $d(V, Q(R)) =$

$$d_i^{j,k} = p_i^{j,k} * |R| - q_i^{j,k} * |V| + (1 - q_i^{j,k}) * |V|$$

Let the attribute A_i and values j, k be the ones that minimizes $d_i^{j,k}$ above over all $1 \leq i \leq n, 1 \leq j \leq k \leq m_i$. Let this value $d_i^{j,k}$ be $d^{\square}_{min}$.

Statistics for a given range $[a, b]$ can be simply computed using the difference of two sums $\sum_{k=1}^{b} p_i^k - \sum_{k=1}^{a-1} p_i^k$. The values $\sum_{k=1}^{j} p_i^k$ can be computed for all j in one pass for each i, i.e., after collecting statistics p_i^j for each column i, we use one additional pass to sum the statistics. Thus, all the sums for a column can be computed in $O(N)$, if hashing is $O(1)$. Thus the necessary statistics (i.e., the sums) are collected in $O(MN)$. However, we need to compute $d_i^{j,k}$, for all i, j, k, which takes $O(MN^2)$ to obtain $d^{\square}_{min}$.

The BVD query corresponds to the Q with the minimum value d' out of $d_{min}, d^{\neq}_{\min}, d^{\square}_{min}$.

THEOREM A.2. *Given a relation instance R and a view instance V, the procedure above finds the EVD/BVD/AVD over queries in $\mathcal{CQ}^1$ in $O(MN^2)$ overall. If $\mathcal{CQ}^1$ does not contain range queries, then the procedure is $O(MN)$.*

This entry is listed in Table 3 in the first 3 columns of the second row.

```
Require: $b_1, b_2, \ldots, b_n \leftarrow$ cardinalities of attributes in view, in decreasing order
Require: $m \leftarrow$ cardinality attributes in relation
 1: $lowest \leftarrow -1$
 2: $best \leftarrow 0$
 3: $product \leftarrow 1$
 4: for $i \in 1 \ldots n$ do
 5:     $product \leftarrow product \times b_i$
 6:     if $lowest > m^i - 2 \cdot product$ then
 7:         $best \leftarrow i$
 8:         $lowest \leftarrow m^i - 2 \cdot product$
 9:     end if
10: end for
11: $conjunct \leftarrow \emptyset$
12: for $i \in best..n$ do
13:     pick any value $v$ from $\mathcal{B}_i$
14:     $conjunct \leftarrow conjunct \wedge v$
15: end for
16: return $conjunct$
```

Algorithm 4: Algorithm to solve the BVD for the special case of R and V being cross-products of sets of values for attributes.

B. FAMILY OF CQ

B.1 EVD

Proof of Theorem 3.3: Let us assume that an EVD query $Q \in \mathcal{CQ}_=$ exists. If the cardinality of attribute A_i in the view V is 1 (i.e., $A_i = v_i$ for some v_i for all tuples in V), then we can add $A_i = v_i$ as a predicate to Q (i.e., $Q' \leftarrow Q \wedge (A_i = v_i)$), if it is not already present, without changing $Q(R)$. Also note that if cardinality of A_i in $V > 1$, then Q cannot have $A_i = v$ for any v, because then $Q(R) \neq V$ since some tuples in V are not selected by Q. Thus any EVD Q can be converted into one of the form $Q' = \wedge_i(A_i = v_i)$ over all A_i that have cardinality 1 in the view. In Algorithm 2, the constructed query Q is precisely Q' above. Now all that is remaining is to check if $Q'(R) = V$ for existence of an EVD, because any other EVD Q, if it exists, can be converted into Q' with $Q(R) = Q'(R)$.

Checking if each attribute is a constant in the view can be done in $O(N)$ and for all attributes in $O(MN)$. Computing $Q(R)$ is $O(MN)$. Thus overall, we have $O(MN)$ complexity.□

B.2 BVD for CQ: Cross Product Special Case

Next we examine the special case when both the relation R and the view V are formed as cross products of sets of attributes. For example, the relation could be all tuples of the form $\{1,2\} \times \{1,2\}$, and the view could be the tuples $\{1,2\} \times \{1\}$.

Formally, let $A_1, A_2, \ldots, A_n$ be the attributes of R. If A_i takes values from the set $\mathcal{A}_i$, then $R = \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \times \mathcal{A}_n$. Additionally, we assume that the view is a cross product as well: $V = \mathcal{B}_1 \times \mathcal{B}_2 \times \ldots \times \mathcal{B}_n$, where A_i in the view takes its values from the set $\mathcal{B}_i \subseteq \mathcal{A}_i$. Additionally, assume that $|\mathcal{A}_1| = |\mathcal{A}_2| = \ldots = |\mathcal{A}_n| = m$. Let $|\mathcal{B}_i| = b_i$. WLOG, assume that $b_1 \geq b_2 \geq \ldots \geq b_n$. We have the following theorem:

THEOREM B.1. *If R and V are both cross products as described above, and if the size of the sets $\mathcal{A}_i$ are all equal to m, then Algorithm 4 returns a solution to the BVD for $\mathcal{CQ}_=$ in $O(MN)$.*

PROOF. In a conjunctive query Q, for each attribute A_i, there are two possible cases: either the attribute is given a value, or the attribute is not present in the conjunctive clause. If the attribute is given a value v, we let $\mathcal{C}_i = \{v\}$, else let $\mathcal{C}_i = \mathcal{A}_i$. Clearly, $Q(R) = \mathcal{C}_1 \times \mathcal{C}_2 \times \ldots \times \mathcal{C}_n$.

For each attribute A_i, consider three numbers f_i, g_i and h_i, where $f_i = |\mathcal{B}_i - \mathcal{C}_i|$, $g_i = |\mathcal{C}_i - \mathcal{B}_i|$, and $h_i = |\mathcal{B}_i \cap \mathcal{C}_i|$.

The difference $d(V, Q(R))$ would then be the following:

$$\begin{aligned} d &= extra + missing \\ &= (\Pi_i(f_i + h_i) - \Pi_i h_i) + (\Pi_i(g_i + h_i) - \Pi_i h_i) \\ &= c + \Pi_i(g_i + h_i) - 2\Pi_i h_i \end{aligned}$$

where c is a constant independent of the conjunctive clause chosen.

We now wish to find the optimal set of g_i, h_i to minimize the number above. Clearly, for each attribute i, we either have (1) $g_i + h_i = m$, and $h_i = b_i$ (2) $g_i + h_i = 1$, and $h_i = 1$.

If we choose k attributes to give values to in the conjunctive clause, $\Pi_i(g_i + h_i)$ will be m^k, but $\Pi_i h_i$ will be atmost $b_1 \cdot b_2 \cdot \ldots \cdot b_k$. Thus our candidates for the best conjunctive clause are precisely those which select a value (1) for A_n from $\mathcal{B}_n$, or (2) for A_n from $\mathcal{B}_n$ and for A_{n-1} from $\mathcal{B}_{n-1}$, or (3) ..., or (n) for A_n from $\mathcal{B}_n$ and for A_{n-1} from $\mathcal{B}_{n-1}$ and ... and for A_1 from $\mathcal{B}_1$. These are precisely the conjunctive clauses considered in Algorithm 4. □

C. UCQ

Proof of Theorem 4.1: If $V \not\subseteq R$, then there is no query that can select the tuples in $V - R$, and hence no EVD. Now, let $V \subseteq R$. Let tuples $t_1, t_2, \ldots, t_n$ be the only tuples in the view, where $t_i = (a_{i1}, a_{i2}, \ldots, a_{im})$. Consider the query corresponding to the selection condition: $\vee_{i=1}^{n}(A_1 = a_{i1} \wedge \ldots \wedge A_m = a_{im})$. This query selects only the tuples $t_1, t_2, \ldots, t_n$ and none other.□

D. GREEDYKUCQ

Proof of Theorem 5.5:
(1) Construct an input such that no conjunctive query has $gb \geq 0$, therefore GREEDYKUCQ doesn't pick any conjunctive query, which is equivalent to the predicate *false*. Therefore, $gb_{greedy} = 0$. However, suppose there are at least k conjunctive queries $Q_1, \ldots, Q_k$, each with $(\frac{N}{k} - 1)$ good elements and $\frac{N}{k}$ bad elements. Suppose for $i \neq j$, Q_i and Q_j don't share any good elements. Further, suppose all Q_i have the identical set of $\frac{N}{k}$ bad elements. Then, by picking these k conjunctive queries we get a total of $(N - k)$ good elements and $\frac{N}{k}$ bad elements, thus $gb_{opt} \geq (N(1 - \frac{1}{k}) - k)$, giving the required result.
(2) Construct an input such that there exists one special conjunctive query Q^* with N good tuples and $(N - \frac{N}{k})$ bad tuples. That is, Q^* covers all elements in V_d but also has $N - \frac{N}{k}$ elements in $R_d - V_d$. In addition, consider k conjunctive queries $Q_1, \ldots, Q_k$, each consisting of $\frac{N}{k}$ good tuples and 1 bad tuple. Further, let the k sets of good tuples be completely disjoint, and this cover V_d completely, and let all of $Q_1, \ldots, Q_k$ have the same one bad tuple. In the construction above:

$$\forall i : gb(Q^*) = \frac{N}{k} > gb(Q_i) = (\frac{N}{k} - 1)$$

Therefore, the greedy solution picks Q^* and stops after that, achieving $d_{greedy} = (N - \frac{N}{k})$, where as $d_{opt} = 1$. Therefore, $r_d = N(1 - \frac{1}{k})$. □

E. BVD-SUP

E.1 CQs with a single predicate

In this section, we describe how to find the BVD-Sup for the family of queries $\mathcal{CQ}^1_=$. For any attribute A_i that is a constant in the view V, let the domain $\mathcal{A}_i$ be $\{1, 2, \dots m_i\}$. Let the proportion of tuples with value j for attribute A_i be p_i^j in the relation R. Let $A_i = j$ for all tuples in the view. Then, if we were to pick $A_i = j$ as the conjunctive clause Q, then $d(V, Q(R))$ would be:

$$d_i = p_i^j * |R| - |V|$$

We simply pick the conjunctive clause corresponding to attribute A_i that minimizes d_i above. Let this value d_i be d_{min}. If there is no such conjunctive clause, we return the conjunctive clause *true*. We have the following theorem:

THEOREM E.1. *Given a relation instance R and a view instance V, the procedure above finds the BVD-Sup over queries in $\mathcal{CQ}^1_=$ in $O(MN)$.*

PROOF. Since all queries in $\mathcal{CQ}^1_=$ have atmost one attribute given one value, we need to consider all single-attribute queries that are supersets of the view V. If an attribute A_i has more than one value in the view V, then no conjunctive clause of the form $A_i = v_i$ would be permissible, because they would not include all the tuples in the view V. Thus we can only consider queries formed from attributes that are constants in the view V. Those are precisely the queries considered above.

The procedure above makes one pass of each column to check if the attribute values are the same $O(MN)$, plus one pass per attribute to compute p_i^j, $O(MN)$, thus we have a time complexity of $O(MN)$. □

E.2 CQs with multiple predicates

The results in this section assume that the size is the same for all conjunctive queries.

We find the solution to the BVD-Sup problem for queries in $\mathcal{CQ}_=$ by picking all attributes that have a single value in the view V, and taking their conjunction. That is, if $A_i = v_i$ for all tuples in the view V, we add $(A_i = v_i)$ as a predicate to the conjunctive query. We add all such predicates to the BVD-Sup query Q.

THEOREM E.2. *Given a relation instance R and a view instance V, the procedure above finds the BVD-Sup over queries in $\mathcal{CQ}_=$ in $O(MN)$.*

PROOF. Let the best query be Q. We can only make Q "better" by adding additional predicates (thereby constraining it further) as long as we maintain $V \subseteq Q(R)$. Thus, if we add the predicate $A_i = v_i$ for all i where A_i has a single value v_i for all tuples in the view, then we make Q only better. Also note that no predicate corresponding to A_j for which there are multiple values in the tuples in the view V can appear in Q, because otherwise $V \subseteq Q(R)$ would be violated (since some tuples would be missed). Thus the best query will contain $A_i = v_i$ for all attributes with a single value in the V, and no predicates corresponding to other attributes, since all attributes are covered, and since a conjunctive clause can have at most one predicate per attribute.

The procedure above makes one pass of each column to check if the attribute values are the same, so we have a complexity of $O(MN)$. □

For the BVD-Sup queries in $\mathcal{CQ}$, we form a conjunctive clause with one range corresponding to each attribute, where the range is between the values that are the minimum and maximum values for the given attribute in the view.

THEOREM E.3. *Given a relation instance R and a view instance V, the procedure above finds the BVD-Sup over queries in $\mathcal{CQ}_=$ in $O(MN)$.*

PROOF. Trivially, the conjunctive clause described above covers all the tuples in the view V. We need to prove that there is no other query that is smaller that covers all the tuples in the view V. If in a new "smaller" query if there is an attribute that is assigned a range that excludes some elements from the range [min, max], then it would be missing some tuples from V. Thus the query selected above is the best possible.

The procedure above makes one pass of each column to find the min and the max attribute values for each attribute, so we have a complexity of $O(MN)$. □

F. BVD-SUB: SINGLE PREDICATE CQ

We now describe the procedure to solve the BVD-Sub problem for $\mathcal{CQ}_=$ in detail. First let us characterize the kind of queries in $\mathcal{CQ}_=$ that could be potential solutions of the BVD-Sub problem.

We can only pick conjunctive clauses of the form $A_i = j$ where the number of tuples in V with value j for attribute A_i is the same as the number of tuples in R with value j for A_i. For all such conjunctive clauses, we compute the difference d, as follows:

For any attribute A_i, let the domain $\mathcal{A}_i$ be $\{1, 2, \dots m_i\}$. Let the proportion of tuples with value j for attribute A_i be p_i^j in the relation R and q_i^j in the view V. Note that $p_i^j * |R| = q_i^j * |V|$, using our condition above. Then, if we were to pick $A_i = j$ as the conjunctive clause Q, then $d(V, Q(R))$ would be:

$$d_i^j = (1 - q_i^j) * |V|$$

We simply pick the conjunctive clause corresponding to the attribute A_i being set the value j that minimizes d_i^j above. Let this value d_i^j be d_{min}.

THEOREM F.1. *Given a relation instance R and a view instance V, the procedure above finds the BVD-Sub over queries in $\mathcal{CQ}^1_=$ in $O(MN)$.*

PROOF. The only conjunctive clauses that are candidate solutions for the BVD-Sub problem are those for which $Q(R) \subseteq V$. Thus the query Q does not contain any tuples not in V. Such queries $(A_i = v_i)$ in $\mathcal{CQ}^1_=$ are precisely the ones for which there are no extraneous tuples corresponding to A_i having value v_i in the relation but not present in the view V. Thus the queries should have the same number of tuples in the view V with value j as are present in the relation R. These are the queries we consider above.

The procedure above makes 1 pass of each column to compute q_i^j for each attribute value, and 1 pass also making sure that the number of tuples in V and R for each attribute value is the same. All of these statistics can be computed in $O(MN)$ using hashing. □