Size and Treewidth Bounds for Conjunctive Queries

Georg Gottlob[*]
Computing Laboratory and
Oxford Man Institute of
Quantitative Finance,
University of Oxford,UK
ggottlob@gmail.com

Stephanie Tien Lee
Computing Laboratory
University of Oxford, UK
stlee@post.harvard.edu

Gregory J. Valiant[†]
Computer Science Division
UC Berkeley
Berkeley, CA, USA
gvaliant@cs.berkeley.edu

ABSTRACT

This paper provides new worst-case bounds for the size and treewith of the result $Q(D)$ of a conjunctive query Q to a database D. We derive bounds for the result size $|Q(D)|$ in terms of structural properties of Q, both in the absence and in the presence of keys and functional dependencies. These bounds are based on a novel "coloring" of the query variables that associates a *coloring number* $C(Q)$ to each query Q. Using this coloring number, we derive tight bounds for the size of $Q(D)$ in case (i) no functional dependencies or keys are specified, and (ii) simple (one-attribute) keys are given. These results generalize recent size-bounds for join queries obtained by Atserias, Grohe, and Marx (FOCS 2008). An extension of our coloring technique also gives a lower bound for $|Q(D)|$ in the general setting of a query with arbitrary functional dependencies. Our new coloring scheme also allows us to precisely characterize (both in the absence of keys and with simple keys) the treewidth-preserving queries—the queries for which the output treewidth is bounded by a function of the input treewidth. Finally we characterize the queries that preserve the sparsity of the input in the general setting with arbitrary functional dependencies.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—relational databases, query processing; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms

Algorithm Design, Theory

[*]Georg Gottlob's work was supported by EPSRC grant EP/E010865/1 "Schema Mappings and Automated Services for Data Integration and Exchange". Gottlob also gratefully acknowledges a Royal Society Wolfson Research Merit Award.
[†]Gregory Valiant's work is supported under a National Science Foundation Graduate Research Fellowship.

Keywords

Database Theory, Conjunctive Queries, Size Bounds, Treewidth

1. INTRODUCTION

In this paper we deal with the general question of how large and intricate the result of a conjunctive query can be, relative to the input relations. We derive worst-case size bounds for the result that depend only on structural properties of the query itself. We assess the intricacy of a data relation via measures of sparsity and treewidth. We derive bounds for the sparsity and treewidth of the result of a query and characterize those conjunctive queries that are sparsity-preserving as well as those that are treewidth-preserving.

Conjunctive queries are the most fundamental and most widely used database queries [7, 20, 1] . They correspond to project-select-join queries in the relational algebra, and to SQL queries of the form `SELECT` ... `FROM` ... `WHERE` ..., whose where-clause equates pairs of attributes of the relations mentioned in the from-clause. Conjunctive queries also correspond to nonrecursive datalog rules of the form

$$R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m),$$

where R_i is a relation name of the underlying database D, R_0 is the output relation, and where each argument u_i is a list of $|u_i|$ variables, where $|u_i|$ is the arity of the corresponding relation, and where the same variable can occur multiple times in one or more argument lists. We allow a single relation R_i to appear several times in the query, thus $m \geq n$. Throughout this paper we adopt this datalog rule representation for conjunctive queries.

It is obvious that, in general, the result of a conjunctive query can be exponentially large in the input size. Even in the case of bounded arities, the result can be substantially larger than the input relations. In fact, in the worst case, the output size is r^k, where r is the size of the largest input relation and k is the arity of the output relation. Queries with very large outputs are sometimes unavoidable, but in most cases they are either ill-posed or anyway undesirable, as they can be disruptive to a multi-user DBMS. It is thus useful to recognize such queries, whenever possible. Obtaining good worst-case bounds for conjunctive queries is, moreover, relevant to view management [20] and data integration [19, 20], as well as to data exchange [11, 18], where data is transferred from a source database to a target database according to schema mappings that are specified via conjunctive queries. In this latter context, good bounds on the result size of a conjunctive query may be used for estimating the amount of data that needs to be materialized at the target site.

In the area of query optimization, models for predicting the size of the output of a conjunctive query based on selectivity indices for relational operators have been developed [24, 17, 8]. The selectivity indices are obtained via sampling techniques (see, e.g. [22, 16]) from existing database instances. Worst case bounds may be obtained by setting each selectivity index to 1, thus assuming the maximum selectivity for each operator. Unfortunately, the resulting bounds are then often trivial (akin to the above r^k bound).

A new and very interesting characterization of the worst-case output size of *join queries* was very recently developed by Atserias, Grohe, and Marx [5]. The join queries form a proper subclass of the conjunctive queries, and correspond to those conjunctive queries in which *all* query variables appear in the head atom $R_0(u_0)$. Their result is based on the notion of *fractional edge cover* [15], and the associated concept of *fractional edge-cover number* $\rho^*(Q)$ of a join query Q. In particular, in [15] it was shown that

$$|Q(D)| \leq \text{rmax}(Q, D)^{\rho^*(Q)}, \tag{1}$$

where $\text{rmax}(Q, D)$ represents the size of the largest relation among $R_1, \ldots, R_n$ in D. In [5] it was shown that this bound is essentially tight.

A substantial part of the present paper deals with the question of whether it is possible to find similar tight worst-case bounds for the output of general conjunctive queries. In particular, we asked this question in three different settings:

1. for general conjunctive queries without integrity constraints;
2. for general conjunctive queries where simple keys have been specified on the underlying database D; and
3. for general conjunctive queries in the setting where arbitrary functional dependencies have been specified on D.

We are able to derive tight bounds similar to those in Formula 1 for settings 1 and 2, and we can give a lower bound for the third (most general) setting. Moreover, for the third setting, we give a precise characterization of the *sparsity preserving* queries - those queries for which in each input database D with domain $\mathcal{U}_D$, the quotient $|R_0|/|\mathcal{U}_D|$ for the output relation R_0 is at most a constant factor larger than the quotient $|D|/|\mathcal{U}_D|$.

A crucial step towards our results is the introduction of a new coloring scheme for query variables, and, accordingly, the association of a *color number* $C(Q)$ with each query Q. Roughly, a valid coloring assigns a set $\mathcal{L}(X)$ of colors to each query variable X and requires that for each functional dependency $V \rightarrow W$, the colors of W are contained in those of V. The *color number* $C(Q)$ of Q is the maximum over all valid colorings of Q of the quotient of the number of colors appearing in the output (i.e., head) variables of Q by the maximum number of colors appearing in the variables of any input (i.e., body) atom of Q.

By using the coloring number $C(Q)$ instead of $\rho^*(Q)$, and by exploiting linear programming duality as in [5], we slightly generalize results of [5] by obtaining the following essentially tight worst-case bounds for the output size of general queries of the form $R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m)$ without specified constraints (Setting 1):

$$|Q(D)| \leq \text{rmax}(Q, D)^{C(Q)},$$

As a corollary we obtain a characterization of the sparsity-preserving queries: A query Q is sparsity-preserving iff $C(Q) = 1$.

It turns out that the color number is equally useful in the case that simple keys are given on the underlying database (Setting 2). However, rather than coloring the original query Q, we color *chase*(Q), which is the result of chasing the keys over the query. The Chase [2, 21, 6, 10, 11] is a well-known procedure for enforcing functional dependencies, and, in particular, keys. For example, let Q be $r(X, Y, Z) \leftarrow s(X, Y) \wedge s(X, Z)$, where the first attribute of the relation s is a key for s, or equivalently, where the functional dependency $s[1] \rightarrow s[2]$ has been specified, then *chase*(Q) is the query $r(X, Y, Y) \leftarrow s(X, Y)$.

We prove the following essentially tight worst-case bounds for the case with simple keys:

$$|Q(D)| \leq \text{rmax}(Q, D)^{C(chase(Q))}.$$

In case of simple keys (or even functional dependencies with single-attribute left sides) the color number $C(chase(Q))$ can be computed in polynomial time via a polynomial-time chase followed by the solution of a polynomially sized linear program. As a corollary, we show that when $C(chase(Q))$ is bounded, and when *each* query variable occurs in the output variables u_0, we immediately obtain a polynomial algorithm for query evaluation.

For composite keys and arbitrary functional dependencies (Setting 3), we derive the same lower bound as for simple keys. While we currently do not know whether the upper bound $\text{rmax}(Q, D)^{C(chase(Q))}$ carries over to this more general setting, we are able to characterize the sparsity-preserving queries. A query Q in this setting is sparsity-preserving iff $C(chase(Q)) = 1$.

An important measure for describing the inherent intricacy of a graph or finite structure is *treewidth* [23]. The treewidth $tw(D)$ of a structure D corresponds in a precise sense to its degree of cyclicity. For example, each tree has treewidth 1, each cycle has treewidth 2, the clique K_k of k elements has treewidth $k - 1$.

By Courcelle's theorem [9], all Boolean queries that can be expressed in terms of monadic second-order logic can be answered in linear time on structures (databases) of bounded treewidth. This result was actually generalized to a large class of queries based on monadic *optimization problems* [4]. By these results, interesting queries that cannot be formulated in SQL or first order logic, and that are NP-hard on arbitrary structures, can be answered in linear time on structures of bounded treewidth. Queries in this class include, for example, questions related to network multicuts [12], program flow graphs [25], or logic-based diagnosis [14]. One may not always issue such queries directly to the original database, but maybe to a view that has been defined via a conjunctive query.

In this context, we posed ourselves the following questions:

- Given a set of relations D of bounded treewidth and a conjunctive query Q. How large is the treewidth of $Q(D)$?
- Is it possible to characterize the queries that *preserve bounded treewidth* in the sense that $tw(Q(D)) = O(tw(D))$?

A join expression $R \bowtie_{A=B} S$ is a *keyed join* if B is a key for S. We first observed that keyed joins are most beneficial for treewidth preservation. (Here B may be a composite attribute.) We derived the following bound on the treewidth of the result of a keyed join $R \bowtie_{A=B} S$ where R has arity k, and S has arity j, and $tw(\langle R, S\rangle) = \omega$ and B is a key of S:

$$tw(R \bowtie_{A=B} S) \leq (j-1)(\omega+1) - 1.$$

Via a nontrivial example we show that this bound is tight up to a small constant factor. We also extend this bound to sequences of keyed joins and to conjunctive queries whose bodies consist of such sequences.

For the case of (fixed) queries Q over (variable) input databases D without keys, we use colorings to precisely characterize the treewith-preserving queries:

> *$tw(Q(D)) = O(tw(D))$ iff $tw(Q(D)) \leq tw(D)$ iff there exists no valid coloring of Q with two colors having color number two.*

We also consider the case where there are arbitrary keys that are not necessarily part of the join attribute. This case can be reduced to the previous case by chasing the keys. We characterize the queries which are treewidth-preserving in this setting, too.

This paper is organized as follows. In Section 2 we state some useful definitions. In Section 3 we define the basic coloring scheme and the color number of a query. In later sections, we extended the basic coloring scheme to deal with keys, and ultimately, to general functional dependencies. Section 4 reports about our size-bounds both without keys and with simple keys, and also deals with sparsity-preservation for those cases. Section 5 presents all our results on treewidth. Section 6 deals with the extension of the work of Section 4 to the setting where arbitrary functional dependencies are specified on the underlying database. Some open problems and topics for future research are stated at the end of Section 6.

2. PRELIMINARIES AND BASIC DEFINITIONS

We assume the reader to be familiar with the theory of relational database systems, relational algebra, and conjunctive queries. As already said in the Introduction, a *conjunctive query* has the form $R(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m)$, where each u_i is a list of (not necessarily distinct) variables of length $|u_i| = arity(R_i)$. Each variable occurring in the query head $R_0(u_0)$ must also occur in the body of the query. The set of all variables occurring in Q is denoted by $var(Q)$. It is important to recall that a single relation R_i might appear several times in the query, and thus m could be larger than n. A *finite structure* or *database* $D = (\mathcal{U}_D, R_1, \ldots, R_k)$ consists of a finite universe $\mathcal{U}_D$ and relations $R_1, \ldots, R_k$ over $\mathcal{U}_D$. The answer $Q(D)$ of query Q over database D consists of the structure $(\mathcal{U}_D, R_0)$ whose unique relation R_0 contains precisely all tuples $\theta(u_0)$ such that $\theta : var(Q) \rightarrow \mathcal{U}_D$ is a substitution such that for each atom $R_i(u_j)$ appearing in the query body, $\theta(u_j) \in R_i$. For ease of notation, we define $\mathrm{rmax}(Q, D)$ to be the number of tuples in the largest relation among $R_1, \ldots, R_n$ in D.

A *(simple) attribute* of a relation R identifies a column of R. An *attribute list* consists of a list (without repetition) of attributes of a relation R. A *compound attribute* is an attribute list with at least two attributes. A list consisting of a unique attribute A is identified with A. The list of all attributes of R is denoted by $attr(R)$. If V is a list of attributes of R and $t \in R$ a tuple of R, then the V-value of t, denoted by $t[V]$ consists of the tuple obtained as the ordered list of all values in V-positions of t.

If V and W are (possibly compound) attributes of R, then a *functional dependency (FD)* $V \rightarrow W$ on relation R expresses that for each $t, t' \in R$, $t[V] = t'[V]$ implies that $t[W] = t'[W]$. Thus each functional dependency $V \rightarrow W$ is equivalent to a set containing a FD $V \rightarrow A$ for each element A of W. If A and B are single attributes, then the FD $A \rightarrow B$ is called a *simple FD*. A (possibly compound) attribute K of R is a *key* iff $K \rightarrow attr(R)$ holds. Such a key is called a *simple key* if K is a simple attribute, otherwise it is called a *compound key*.[1] An argument position in an atom that corresponds to a simple key attribute is referred to as a *keyed position*.

Let $D = (\mathcal{U}_D, R_1, \ldots, R_k)$ be a structure. The *Gaifman graph* $G(D)$ of D is the graph $(\mathcal{U}_D, E)$, where $\{a, b\} \in E$ iff a and b are two distinct elements from $\mathcal{U}_D$ who appear jointly in some tuple of D.

Given a graph $\mathcal{G} = (V, E)$, a *tree decomposition* [23] of $\mathcal{G}$ is a pair (T, λ), where $T = (N, A)$ is a tree, and λ a labeling function $\lambda : N \rightarrow 2^V$ such that: *(i)* for all $v \in V$, there exists $n \in N$ such that $v \in \lambda(n)$; *(ii)* for all edges $e \in E$, there exists $n \in N$ such that $\lambda(n) \supseteq e$; *(iii)* for every $v \in V$, the set $\{n \in N \mid v \in \lambda(n)\}$ induces a connected subtree in T.

The *width* of a tree decomposition (T, λ) is the integer value $\max\{|\lambda(n)| - 1 \mid n \in N\}$. The *treewidth* of a graph $\mathcal{G} = (V, E)$, denoted $\mathrm{tw}(\mathcal{G})$, is the minimum width of all tree decompositions. Given a finite structure $D = (\mathcal{U}_D, R_1, \ldots, R_k)$, the treewidth of D is defined to be the treewidth of its Gaifman graph, formally $\mathrm{tw}(D) = \mathrm{tw}(G(D))$.

The following motivating example shows that the size and treewidth of the result of a conjunctive query over a database D can be significantly larger than the size and treewidth of D, respectively.

Example 2.1. *Consider the relation*

$$R(A, B) = \{\langle 1, 1\rangle, \langle 1, 2\rangle, \ldots, \langle 1, n\rangle\}$$

that has treewidth 1. The relation $R' = R \bowtie_{A=A} R$, which is equivalently obtained as the result of the conjunctive query

$$R'(X, Y, Z) \leftarrow R(X, Y) \wedge R(X, Z),$$

has n^2 tuples where each of the n elements appears in a tuple with each of the other elements. Thus the Gaifman graph is the complete graph on n vertices, which has treewidth $n-1$. Letting n get large, we see that the treewidth of R' can be arbitrarily large, while the treewidth of R is still 1.

Definition 2.2. *For a database D, we define the* sparsity *of D to be $sp(D) := |D|/|\mathcal{U}_D|$, where $|D|$ is the number of tuples in D, and $|\mathcal{U}_D|$ is the number of elements in the domain of D.*

Remark If we are to consider the blowup of the sparsity after the application of a query Q, one could decide whether to define $\mathcal{U}_{Q(D)}$ to be the same as $\mathcal{U}_D$, or whether we allow $\mathcal{U}_{Q(D)}$ to be restricted to the *active domain* of $Q(D)$, i.e., to those domain values that occur effectively in $Q(D)$. In the

[1]Note: We do not require compound keys to be minimal.

second case, even without increasing the number of tuples, by a single projection operation we can increase the sparsity (by, for example, removing all the key attributes). Even for a single keyed join, for any N, one can easily construct a relation R with $\mathrm{sp}(R) = 1$ such that $\mathrm{sp}(R \bowtie R) = N$. For this reason, in the remainder of this section our analysis will focus on the richer and more interesting problem of understanding the blowup in sparsity using the first definition, where we fix $\mathcal{U}_{Q(D)}$ to be the same as $\mathcal{U}_D$.

There is also a question of how to define the sparsity of a query input list $\langle R_1, \ldots, R_n \rangle$ corresponding to a query body $R_1(u_1) \wedge \ldots \wedge R_n(u_m)$, given that this may contain repeated occurrences of the same relation symbol. For convenience we define the sparsity to be $\sum_{j=1}^{m} |R_{i(j)}|/|\mathcal{U}_D|$, that is, we double-count relations that occur multiple times. If we were to only take $\sum_{i=1}^{n} |R_i|/|\mathcal{U}_D|$ then one could concatenate all the R_i's to form a relation R with arity the sum of the arities of the R_i's, and $|R| = \max_i |R_i|$, such that the result of applying the corresponding query with each R_i replaced by R is unchanged, but the sparsity would be smaller.

Definition 2.3. *A query Q over input relation list $\langle R_1, \ldots, R_n \rangle$ (possibly with repeated relations) is sparsity preserving if for every database instance,*

$$sp(Q(\langle R_1, \ldots, R_n \rangle)) = O(sp(\langle R_1, \ldots, R_n \rangle)).$$

3. THE COLORING

In this section we define a coloring scheme for assigning colors to the variables appearing in a conjunctive query. This coloring underlies both our tight bounds on the size of the results of the query, and our characterizations of queries that have bounded sparsity and bounded treewidth. We start by giving the basic definition that applies to the setting without any functional dependencies. In section 4.2 we extend this definition to accommodate simple keys, and finally in section 6 we further extend this definition in a natural way to the case where general functional dependencies are specified.

Definition 3.1. *Given a conjunctive query $Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m)$, without any functional dependencies a* valid coloring *of Q with c colors assigns to each variable X a label $\mathcal{L}(X) \subseteq \{1, \ldots, c\}$ consisting of zero or more colors.*

Definition 3.2. *The* color number *of a query $Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m)$, denoted $C(Q)$, is the maximum over valid colorings of Q of the ratio of the total number of colors appearing in the output variables u_0, to the maximum number of colors appearing in any given u_i, for $i \geq 1$. Formally:*

$$C(Q) := \max_{colorings} \frac{|\cup_{X_i \in u_0} \mathcal{L}(X_i)|}{\max_{i \geq 1} |\cup_{X_j \in u_i} \mathcal{L}(X_j)|}.$$

Example 3.3. *Let Q be the query*

$$S(X, Y, Z) \leftarrow R(X, Y) \wedge R(X, Z) \wedge R(Y, Z).$$

Assume no keys are asserted. Then $C(Q) = 3/2$, which is attained with three different colors for the variables X, Y, and Z.

4. SIZE BOUNDS AND SPARSITY

In this section we present our main results regarding size bounds and sparsity. We consider conjunctive queries without keys and with simple keys. Whenever we use the word "key" in this section we mean "simple key". Essentially, the color number of a query will be a tight bound on the exponent relating the sizes of the input and the output database. We start this section by considering the special case in which the given query has no keyed relations, and relate our characterization to previous results regarding size bounds. We will then reduce the general case with keys to this special case.

4.1 Size Bounds Without Keys

We begin by considering the case when no relations have keys. While this case has been considered in [15, 5], our approach will allow us to extend the results to the case with specified output variables and keys.

Proposition 4.1. *Given a query $Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m)$ without specification of FDs or keys, for any database D,*

$$|Q(D)| \leq rmax(Q, D)^{C(Q)},$$

where $rmax(Q, D)$ is the size of the largest relation among $R_1, \ldots, R_n$ in D. Furthermore, this bound is essentially tight: for any integer $N > 0$, there exists a database D with $rmax(Q, D) \leq rep(Q) \cdot N$, and $|Q(D)| = N^{C(Q)}$, where $rep(Q)$ is defined to be the maximum number of times any specific relation R_i appears in Q.

Corollary 4.2. *A query Q, as in Proposition 4.1, without FDs or keys is sparsity preserving if, and only if $C(Q) = 1$. Equivalently, Q is sparsity preserving if, and only if $\exists i \geq 1$ such that the variables in u_0 are a (not necessarily proper) subset of the variables of u_i.*

To prove Proposition 4.1 we first show that $C(Q)$ is the solution to a linear program, then, exploiting duality, reduce it to the upper bound given in [15, 5].

Proof of proposition 4.1: Given a query $R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m)$, without any keys or FDs, note that there is always an optimal coloring with the property that for all $X_i \notin u_0, \mathcal{L}(X_i) = \emptyset$. Next, observe that it suffices to assume that no color appears in the label of more than a single variable. To see this, if a color appears in the labels of at least two variables, by removing it from the label of one of the labels, the numerator of the expression for the color number remains unchanged whereas the denominator can only decrease. Thus the color number of such a query is given by the following expression, where $x_i \in \mathbb{N}$ denotes the number of colors assigned to variable $X_i \in u_0$:

$$\text{maximize } \frac{\sum_{X_i \in u_0} x_i}{\max_{j \geq 1} \sum_{X_i \in u_j} x_i}.$$

Pushing the normalization factor into constraints on the x_i's, the above expression is seen to be equivalent to the following linear program:

$$\begin{array}{ll} \text{maximize} & \sum_{X_i \in u_0} x_i \\ \text{subject to} & \sum_{X_i \in u_j} x_i \leq 1 \quad \forall j \geq 1 \\ & x_i \geq 0. \end{array}$$

It is worth stressing that the above linear program is *not* a proper relaxation of the quantity in question–any rational solution p/q to the above with $x_i{=}r_i/q$ can be transformed into a valid coloring with p colors such that $|\mathcal{L}(X_i)| = r_i$. The constraints of the linear program then imply $\max_j |\cup_{X_i \in u_j} \mathcal{L}(X_i)| \leq q$.

The dual linear program is:

$$\begin{array}{ll} \text{minimize} & \sum_j y_j \\ \text{subject to} & \sum_{u_j : X_i \in u_j} y_j \geq 1 \quad \forall X_i \in u_0 \\ & y_i \geq 0. \end{array}$$

This linear program can be recognized as expressing the minimal fractional edge cover of the hypergraph defined by Q, in which variables that don't appear in u_0 have been removed. Since the size increase of $Q(D)$ is at most that of the analogous query in which variables not appearing in u_0 have been removed, the upper bound now follows from [15], in which Shearer's Lemma is employed to exploit the submodularity of the entropy function to yield a combinatorial statement. (We note that an elementary argument based on the pigeon-hole principle suffices to show that if there is any size increase, $C(Q) > 1$.)

The lower bound is by a simple construction, and is noted in [5]. A generalization of this construction will prove useful in the remainder of this paper, and we explicitly describe it here.

Given any (not necessarily optimal) coloring with d colors and color number $C(Q)$, we construct an instance with the desired size increase as follows. Fix an integer N. For each variable, X_i, we assign $N^{|\mathcal{L}(X_i)|}$ distinct values, which we will denote by
$x^i_{1,\ldots,1}, \ldots, x^i_{N,\ldots,N}$. For each of the N^d possible ways of selecting d numbers from $\{1, \ldots, N\}$ with replacement, we create a corresponding tuples with $|V|$ elements. The mapping is clear: for a variable X_i with $\mathcal{L}(X_i) = \{i_1, \ldots, i_{|\mathcal{L}(X_i)|}\}$, the value taken by X_i in the tuple corresponding to $j_1, \ldots, j_d$ will be $x^i_{j_{i_1}, \ldots, j_{i_{|\mathcal{L}(X_i)|}}}$. Finally, we let R_i be the union of the projections of the final set of tuples onto each u_j where $R_i(u_j)$ occurs in Q.

In the above construction, there will be at most

$$\sum_{i : R_k(u_i) \text{ in } Q} N^{|\cup_{X_j \in u_i} \mathcal{L}(X_j)|}$$

tuples in input relation R_k, and $N^{|\cup_{X_j \in u_0} \mathcal{L}(X_j)|}$ tuples in the output, from which our lower bound on the blowup follows. □

While Proposition 4.1 is similar to the size bounds based on the fractional edge cover number, and coincides with it in case all query variables occur in the head's atom, our characterization in terms of colorings provides a more concrete and intuitive connection with database instances; each color represents a degree of freedom of the output, in a sense illustrated in the above construction. This more intuitive characterization of the fractional edge cover in terms of colorings allows us to extend Proposition 4.1 to the case with simple keys and general conjunctive queries that include a projection. Furthermore, while the color number will not be directly applicable to bounding the possible increase in treewidth, a related property of the colorings will allow us to characterize those queries with bounded blowup in treewidth.

4.2 Size Bounds with Simple Keys

We apply the intuition of the color number provided in the previous section with the hope of yielding a similar result to Proposition 4.1. We start by extending our definition of a valid coloring to the setting with simple keys, and then give an example illustrating that the addition of keys might complicate both the arguments for the upper, as well as lower bounds. Nevertheless, after addressing this slight complication, we will see that the formal statement of Proposition 4.1 applies almost without modification (using the extended definition of color number).

Definition 4.3. *Given a conjunctive query and set of simple keys a* valid coloring *of Q with c colors assigns to each variable X_i a label $\mathcal{L}(X_i) \subseteq \{1, \ldots, c\}$ consisting of zero or more colors such that the following condition is satisfied:*

- *For each atom with X appearing in a keyed position, for all variables Y such that X is a key for Y, we have that $\mathcal{L}(X) \supseteq \mathcal{L}(Y)$.*

The definition of the color number, $C(Q)$, given in the previous section continues to apply in the setting with key information, except with the more restrictive definition of *valid colorings* given here in place of the original Definition 3.1.

Example 4.4. *Consider the query*

$$R_0(ABCD) \leftarrow R_1(ABC) \wedge R_1(AAA) \wedge R_2(CD),$$

together with the first position of R_1 being a key for R_1. The labeling

$$\mathcal{L}(A) = \{1,2,3\},\ \mathcal{L}(B) = \{2\},\ \mathcal{L}(C) = \{3\},\ \mathcal{L}(D) = \{4\},$$

is a valid coloring, yielding $C(Q) = 4/3$. Despite this, the information that the first position of R_1 is a key for R_1, and the atom $R_1(AAA)$ imply that in any output tuple, the values in positions corresponding to $A, B,$ and C must all be equal. Thus in addition to the key information, there is also the further restriction on the output tuples that the actual values in positions B and C must be the same as the value in position A. It follows that there can be at most $|R_2|$ tuples in the output.

The above example motivates the *chase* operation which reduces the set of variables so as to eliminate any implied dependencies on the *values* taken by the variables beyond that implied by the given set of simple keys.

Definition 4.5. *Given a conjunctive query*

$$Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m),$$

we define chase(Q) to be the result of iteratively performing the following replacements:

- *Given two atoms $R_i(u_j)$ and $R_i(u_k)$ of the same relation, with the p^{th} position a key for relation R_i, if the variable at the p^{th} position of u_j is the same as the variable at the p^{th} position of u_k, then for each $h \in 1, \ldots, |u_j|$ let X be the variable that occurs at position h in u_j. We replace every instance of X that occurs anywhere in the query by the variable occurring at position h of u_k, and proceed with the updated u_i's. Finally, we remove the term $R_i(u_j)$ from the conjunctive query.*

While the above definition only applies to queries with simple keys, the chase operator extends to arbitrary functional dependencies, though we defer the reader to [21] for details.

The following fact confirms the intuition that the substitutions in Definition 4.5 do not affect the result of the query.

Fact 4.6. *[21, 3, 6] For any instance, the result of applying the query $chase(Q)$ is identical to the output of applying Q.*

We are now ready to state and prove our main result regarding the potential size increase after applying a query.

Theorem 4.7. *Given a query $Q = R(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m)$ and set of simple keys,*

$$|Q(D)| \leq rmax(Q, D)^{C(chase(Q))}.$$

Furthermore, this bound is essentially tight: for any $N > 0$, there exists a database D with $rmax(Q, D) \leq rep(Q) \cdot N$, and $|Q(D)| = N^{C(Q)}$, where $rep(Q)$ is the maximum number of times any specific relation R_i appears in Q.

First note that the upper bound does not follow trivially from Proposition 4.1 because the color number in the presence of key information may be smaller than if the key information is ignored. To prove the upper bound, we will transform $chase(Q)$ and the set of keys into a query Q' over a modified set of relations that has no keys. This transformation will have the property that $C(chase(Q)) = C(Q')$, and the transformation preserves the relationship between the size of the original structure and the size of the result. To obtain Q', we first let the query Q^* be the same as $chase(Q)$, except have each atom $R_i(u_j)$ refer to a unique relation. Thus Q^* can be written as $R_0(u_0) \leftarrow R'_1(u_1) \wedge \ldots \wedge R'_m(u_m)$. Furthermore, the keyed positions of R'_i are the same as the keyed positions of the original relation R_j where $R_j(u_i)$ was the term in $chase(Q)$ that became $R'_i(u_i)$ in Q^*. Finally, we modify the relations of Q^* by iteratively applying the following insertion until all the keys have been 'removed':

- Given a variable X appearing in a keyed position in a relation R that also contains variables $Y_1, \ldots, Y_k$, we replace all occurrences of X in the entire query with the list of variables $X, Y_1, \ldots, Y_k$, and we remove the property that X is in a keyed position of R.

The following example should clarify the above process.

Example 4.8. *Consider $chase(Q) = Q^* = R_1(ABC) \wedge R_2(AD) \wedge R_3(EA)$, where the first attribute of each relation is a key for the respective relation. Variable A is a key for B, C, in R_1, and thus one iteration of the insertion yields $Q^* = R_1(ABC) \wedge R_2(ABCD) \wedge R_3(EABC)$. Performing the insertion for the variable A which is a key for R_2 yields $R_1(ABCD) \wedge R_2(ABCD) \wedge R_3(EABCD)$, which is unaffected by the final insertion corresponding to E being a key for R_3. Thus*

$$Q' = Q^* = R_1(ABCD) \wedge R_2(ABCD) \wedge R_3(EABCD).$$

We now argue that the above transformation does not alter the color number. The proof of the following lemma is deferred to the extended version of this paper [13].

Lemma 4.9. *The color numbers of $chase(Q)$ and Q' are equal, that is $C(chase(Q)) = C(Q')$.*

We are now ready to prove Theorem 4.7.

Proof of theorem 4.7: We first prove that the lower bound holds by arguing that the construction described in the proof of Proposition 4.1 continues to be valid in the presence of keys. We then argue that the potential increase in size of Q' is at least of Q, from which the upper bound follows from Proposition 4.1 together with Lemma 4.9.

In the case that each relation only appears once, and thus $m = n$, our construction clearly respects every functional dependency. This is because for each atom $R_i(u_i)$ our definition of a valid coloring guarantees that every color appearing in the labels of variables in u_i must appear in its key, and thus for each tuple that is added to R_i in our construction,the indices of the values in the positions of the key uniquely determines the indices of the values in the rest of the positions. Now, consider the case that some relation R_i appears more than once in Q. If both $R_i(u_j)$ and $R_i(u_k)$ appear in Q, we claim that we can union the set of tuples corresponding to u_j and u_k without violating the keyed positions. Assume position 1 is a key for R_i. We are guaranteed by the chase operation that if variable X occurs in position t of u_j, then a variable $Y \neq X$ occurs in position t of u_k. Since each variable has a disjoint set of values which it can take in our construction, we may take the union of the two sets of tuples without generating any duplicate values of the key. It follows that the claimed lower bound holds.

To see that the upper bound holds, let Q_1 be defined, as above, to be the query and set of key attributes. yielded after a single insertion removed the dependency implied by X being a key for $Y_1, \ldots, Y_k$. To see that the size increase of Q_1 is at least that of Q, given an instance of Q_1, we define an instance of Q where both the input size, and resulting number of tuples are the same as for Q_1. For each possible value x in position X there must be a unique set of values $y_1, \ldots, y_k$ in respective positions $Y_1, \ldots, Y_k$, and thus in every tuple of Q that has value x, we may add the extra attributes $Y_1, \ldots, Y_k$, and populate them uniquely with corresponding values $y_1, \ldots, y_k$. This clearly does not change the number of input tuples. Furthermore, even if X is in the output projection u_0, then the above process applied to each output tuple yields the same number of output tuples, and is clearly compatible with the query Q_1. □

As a consequence of the transformation from the case with simple keys to the case without any keys in the above proof, we get the following proposition, whose proof is deferred to the extended version.

Proposition 4.10. *For a conjunctive query Q together with simple key information, the color number, $C(Q)$, can be computed in polynomial time. Equivalently, the maximum possible increase in the size of a database as a result of a conjunctive query with simple keys can be computed in polynomial time.*

Finally, the following corollary to Theorem 4.7 captures the fact that if $C(Q)$ is bounded, not only is the size of $|Q(D)|$ at most polynomial in $|D|$, but, in case all variables are output variables, $Q(D)$ can be computed in polynomial time via a join-project plan. The proof follows easily from the proof of Theorem 4.7 together with Theorem 15 of [5], and we defer it to the extended version.

COROLLARY 4.11. *Given a conjunctive query*

$$Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m),$$

with simple keys, let $Q' = R_0(u_0') \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m)$, where u_0' consists of each variable that occurs in $u_1, u_2, \ldots, u_m$. For any database D, there exists a join-project plan for calculating $Q(D)$ in time

$$O\left(|u_0|^2 \cdot m \cdot rmax(Q, D)^{C(chase(Q'))+1}\right).$$

Furthermore, such a plan can be calculated in time $O(|Q|^2)$.

It is worth noting that the condition that all variables appear in the output is necessary in the above corollary; in the absence of this condition, it is easy to construct examples of databases and queries for which $C(Q) = 1$ but for which evaluating the query is equivalent to solving an NP-hard problem.

5. TREEWIDTH

In this section we consider the problem of deciding whether the result of a general conjunctive query together with a set of keys can have treewidth unbounded by any function of the treewidth of the inputs and the query. While the characterization for bounded treewidth queries will be different than our characterization of sparsity-preserving queries, the characterization will still be based on properties of the set of valid colorings of the query. As in the previous section on size bounds, we will begin by considering queries that have no functional dependencies. We will then reduce the general case to the case without functional dependencies, using a similar strategy to the reduction in the proof of Theorem 4.7 and demonstrate that if our condition for boundedness is satisfied then one can express the result of the query as a subset of a sequence of keyed joins of the original relations together with some extra relations whose treewidth is bounded. The significant complication is that in the proof of Theorem 4.7, we implicitly employed the fact that a keyed join operation preserves the number of tuples; trivial examples illustrate that a keyed join may increase the treewidth.

We begin this section by providing a tight bound on the possible increase in treewidth after a single keyed join, which is of independent interest. We then give our characterization of queries without functional dependencies that induce a bounded increase in treewidth. Finally, we employ the boundedness of treewidth after a keyed join to reduce the general case to the case without functional dependencies.

5.1 Treewidth of Keyed Joins

We present tight bounds on the blowup of the treewidth of the result of a single keyed join operation. The bound follows easily from the definition of treewidth, and the example illustrating the tightness of the bound is a nontrivial construction in which the Gaifman graphs of the initial relations are grids with some extra nodes and edges, and the Gaifman graph of the result contains a quadratically larger grid. We begin with the construction.

The following easily proved fact is helpful to keep in mind:

FACT 5.1. *The treewidth of an $n \times m$ rectangular grid is* $\min(n, m)$.

PROPOSITION 5.2. *For any n, m with $m \leq n-2$, there exists a relation R of arity $m+2$ whose Gaifman graph has treewidth n, such that after a single keyed join operation, $R \bowtie R$, the resulting treewidth is at least nm.*

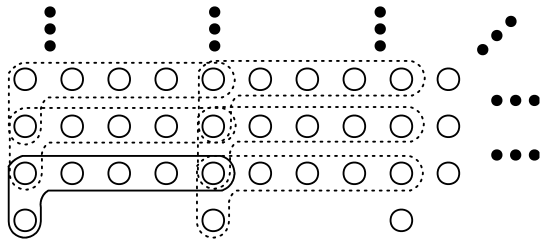

Figure 1: The structure of G in the case $m = 4$, with the partitions into $S_{i,j}$ indicated.

PROOF. Consider an $(nm+1) \times nm$ rectangular lattice, together with n additional vertices $\{\alpha_1, \ldots, \alpha_n\}$. We partition the $O(n^2m^2)$ vertices into sets $S_{i,j}$, for $1 \leq i \leq nm$, and $1 \leq j \leq n$ as follows. For $i = 1$, let $S_{i,j}$ consist of vertices $\alpha_j, v_{1,m(j-1)+1}, v_{1,m(j-1)+2}, \ldots, v_{1,mj+1}$. For $i \geq 2$, let $S_{i,j}$ consist of vertices

$$v_{i-1,m(j-1)+1}, v_{i,m(j-1)+1}, \ldots, v_{i,m(j-1)+m+1}, v_{i-1,mj+1}.$$

For each (i, j) pair, we add edges so as to make $S_{i,j}$ a complete graph. Let G denote the resulting graph. The structure of G is suggested in Figure 1, in which the partition of the nodes into the groups $S_{i,j}$ is depicted.

The proof now follows from the following lemmas. These lemmas are intuitively clear, and we defer their proofs to the extended version. □

LEMMA 5.3. *The treewidth of G is n.*

LEMMA 5.4. *For the relation R depicted in Figure 1, where each node in the graph is taken to be a unique value, the treewidth of the keyed join $R \bowtie_{A_1=A_2} R$, is at least nm, where A_i is the i^{th} position of R.*

The following theorem shows that the construction of Figure 1 yields the worst-case increase in treewidth.

THEOREM 5.5. *Let R, S be relations, where R has arity k, and S has arity j, and $tw(\langle R, S\rangle) = \omega$. If B is a key in S, then,*

$$tw(R \bowtie_{A=B} S) \leq (j-1)(\omega+1) - 1.$$

Furthermore, this bound is tight to a constant factor.

The tightness of the bound follows from Proposition 5.2. The proof of the bound depends on the following easily verified lemma:

LEMMA 5.6. *Let G be a graph with tree decomposition $\mathcal{T} = (\{X_i | i \in I\}, T = (I, F))$. For $x, y \in I$, let $p_{x,y}$ be a unique path between x and y in T. Let $W \subseteq X_y$ and let X_k', for $k \in I$ be defined as follows:*

$$X_k' = \begin{cases} X_k \cup W & \textit{if } k \textit{ lies on } p_{x,y} \\ X_k & \textit{otherwise.} \end{cases}$$

$\mathcal{T}_{(x,y)} = (\{X_i' | i \in I\}, T = (I, F))$ is also a tree decomposition of G.

Proof of Theorem 5.5: The proof is explicitly constructing a tree decomposition for $R \bowtie_{A=B} S$, from a tree decomposition of $\langle R, S\rangle$. Let Y be a tree decomposition of $\langle R, S\rangle$, and t, u tuples such that $t \in R$ and $u \in S$ and $t(A) = u(B)$. There is some bag $X_t \in Y$ that contains all elements of t, and some bag $X_u \in Y$ that contains all elements of u. Since X_u and X_t have the common element $t(A)$, there is a path between X_t and X_u, $X_t = X_1, X_2, ..., X_{m-1}, X_m = X_u$. Let $W = \{v : v \in u, v \neq u(B)\}$, and for each $i = 1, \ldots, m-1$, consider augmenting each bag X_i by adding the elements of W. By Lemma 5.6, this procedure maintains a valid tree decomposition. After performing this augmentation for every pair of tuples u, t that satisfy $t(A) = u(B)$, we clearly have a tree decomposition for $R \bowtie_{A=B} S$, since every tuple in the result appears together in some bag.

We now analyze the size of the largest bag created in the above construction. First observe that since Y is a valid tree decomposition, each bag along the path $X_t = X_1, X_2, ..., X_{m-1}, X_m = X_u$ must contain the value $u(B)$. Consider any bag X, which initially has size at most $\omega + 1$. For each value $v \in X$, there is at most one tuple $t \in S$ such that $t(B) = v$, and thus in our construction, X can be augmented at most $\omega + 1$ times. Since each augmentation increases the size of X by at most $j - 1$, the theorem holds. $\square$

The following intuitively clear proposition extends the above bounds to a sequence of keyed joins.

PROPOSITION 5.7. *Let the query Q be of the form*

$$\left(\ldots((R_1 \bowtie_{A_1=A_2} R_2) \bowtie_{B_1=A_3} R_3) \bowtie \ldots \bowtie_{B_{n-1}=A_n} R_n\right),$$

such that for $i \geq 2$ the attribute A_i occurs in a keyed position of R_i and let $l = \max(\max_i(arity(R_i)), 3)$.

$$tw(Q) \leq l^{n-1} \max(tw(\langle R_1, \ldots, R_n\rangle), 2).$$

The above proposition is unsurprising in light of Theorem 5.5, though the proof is nontrivial and involves some bookkeeping. We defer the proof to the extended version.

5.2 Bounding Treewidth Without Keys

In this section, we present a necessary and sufficient condition for the results of a general conjunctive query without keys to have treewidth bounded by some function of the input treewidth.

PROPOSITION 5.8. *Given a conjunctive query*

$$Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m),$$

with no keys, $tw(Q(D))$ can not be bounded as any function of $tw(\langle R_1, \ldots, R_n\rangle)$, m, and $|u_i|$ if there exists a valid coloring of Q with 2 colors and color number 2. Furthermore, if there is no such coloring, then

$$tw(Q(D)) \leq tw(\langle R_1, \ldots, R_n\rangle).$$

PROOF. We first argue that given a valid coloring with 2 colors and color number 2, the method of populating the relations given in the proof of Proposition 4.1 yields relations with $|R_i| \leq N$, such that $\text{tw}(\langle R_1, \ldots, R_n\rangle) \leq \sum_i |u_i|$ and $\text{tw}(Q(D)) \geq N$. Indeed, for each u_i with $i \geq 1$ $|\cup_{X \in u_i} \mathcal{L}(X)| \leq 1$ so for each of the variables with a label with a color, each value appears in only one tuple, and thus the Gaifman graph corresponding to each atom consists of the clique of the values corresponding to the uncolored variables, with N disjoint cliques attached only via the values corresponding to uncolored nodes. Thus for each atom $R(u_i)$, the corresponding Gaifman graph has treewidth at most $|u_i|$. Finally, if one adds the Gaifman graph corresponding to $R'(u_j)$ to the Gaifman graph corresponding to $R(u_i)$, no new edges will be added between values already in the Gaifman graph of $R(u_i)$, and thus $\text{tw}(\langle R, R'\rangle) \leq |u_i| + |u_j|$, from which our claim follows. In the result of the query, each of the N values taken by a variable with color 1 appears with each of the N values taken by variables with color 2 (since $|\cup_{X \in u_0} \mathcal{L}(X)| = 2$), and thus the treewidth of the result will be at least N, and, in particular, can be made arbitrarily larger than the original treewidth.

For the other direction, if there is no valid coloring with 2 colors and color number 2, it means that for each pair of variables X_i, X_j that appear in the final projection u_0, there must be some atom u_k containing both X_i and X_j. Thus every edge in the Gaifman graph of the result will exist in the Gaifman graph of one of the inputs, and $\text{tw}(Q(D)) \leq \text{tw}(\langle R_1, \ldots, R_n\rangle)$, as claimed. $\square$

5.3 Treewidth and Keys

We now extend the characterization of queries that preserve bounded treewidth to the setting with simple keys via a reduction to the result of the previous section. Our reduction analyzes the iterative process of removing keys, as employed in the proof of Theorem 4.7, together with the bounds of Proposition 5.7 for the increase in treewidth resulting from a keyed join operation.

THEOREM 5.9. *Given a conjunctive query*

$$Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m),$$

and a set of simple keys, $tw(Q(D))$ can not be bounded as any function of $tw(\langle R_1, \ldots, R_n\rangle)$, m, and u_i if there exists a valid coloring of $chase(Q)$ with 2 colors and color number 2. Furthermore, if there is no such coloring, then for any database D,

$$tw(Q(D)) \leq p^{km-1} \max(tw(\langle R_1(D), \ldots, R_n(D)\rangle), 2),$$

where $k = \max_i |u_i|$, and p is the total number of variables appearing in Q.

PROOF. First observe that if there is a valid coloring with 2 colors and color number 2, the construction given in the proof of Proposition 5.8 is still valid, and yields an instance with unbounded blowup in treewidth. For the other direction, we consider the insertion procedure given in the proof of Theorem 4.7. Let $Q = Q_0, Q_1, \ldots, Q_{km} = Q'$ be the queries obtained by sequentially performing insertions, where Q' is as in the proof of Theorem 4.7 and has no keys. (Note that there can be at most km such insertion steps.) A trivial modification to the proof that $C(Q) = C(Q_i) = C(Q')$ demonstrates that Q has a valid coloring with 2 colors and color number 2 if, and only if for all i, Q_i has such a coloring.

Assume for the point of contradiction that there is an instance with

$$\text{tw}(Q(D)) > p^{km-1} \max(tw(\langle R_1(D), \ldots, R_n(D)\rangle), 2).$$

By applying the natural transformation from instances of Q_i to instances of Q_{i+1} given in the proof of Theorem 4.7, for any $i \in 0, \ldots, km-1$, given any instance of Q_i, one can make an instance of Q_{i+1} in which the Gaifman graph of the input structure of Q_{i+1} is contained by the Gaifman graph of $\langle R_1^i, \ldots, R_n^i \rangle \bowtie S$ where the join is keyed, and S is the relation of the form $(X, Y_1, \ldots, Y_j)$ where X is a key for all Y_i. Furthermore, note that

$$\text{tw}\left(\langle R_1^i, \ldots, R_n^i \rangle, S\right) = \text{tw}\left(\langle R_1^i, \ldots, R_n^i \rangle\right),$$

since each tuple of S necessarily appears as a subset of a tuple in some R_j^i. Iterating this argument for $i = 0, \ldots, km-1$, together with Propositions 5.7 and 5.8 yield

$$\text{tw}(Q'(D')) \leq p^{km-1} \max\left(tw(\langle R_1, \ldots, R_n \rangle), 2\right),$$

which contradicts our assumption because $\text{tw}(Q(D)) \leq \text{tw}(Q'(D'))$, where D' is the database instance compatible with Q' obtained from D. $\square$

6. EXTENSIONS TO GENERAL FUNCTIONAL DEPENDENCIES

In this section, we attempt to extend the results of the previous two sections to conjunctive queries with arbitrary functional dependencies. We begin by considering a natural extension of the definition of 'valid colorings' that applies to arbitrary functional dependencies. We then provide partial analogs of Theorems 4.7 and 5.9, whose proofs follow the rough structure of the analogous proofs for simple keys, but with several significant complications.

Consider the following generalization of the definition of *valid colorings* that was given in section 3:

Definition 6.1. *Given a conjunctive query*

$$Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m),$$

and the set of functional dependencies for each input relation, a valid coloring *of Q with c colors assigns to each variable X_i a label $\mathcal{L}(X_i) \subseteq \{1, \ldots, c\}$ consisting of zero or more colors such that the following condition is satisfied:*

- *For each functional dependency $X_1, \ldots, X_k \to Y$,*

$$\mathcal{L}(Y) \subseteq \bigcup_i \mathcal{L}(X_i).$$

The above extension of *valid colorings* implicitly extends our definition of the *color number* of a query and associated set of functional dependencies.

Remark 6.2. *It is worth noting that simple functional dependencies are a strict generalization of simple keys—simple keys can be expressed as (sets of) simple functional dependencies, though the converse is not true. Unsurprisingly, all of our results for queries with simple keys hold (with only minimal modification of the proofs) in the slightly more general setting of simple functional dependencies.*

The following proposition extends the lower bounds of Theorem 4.7 to this general setting. The proof is similar, though considerably more involved in this setting.

Proposition 6.3. *Given a query $Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m)$ and set of functional dependencies, there exists an instance D in which*

$$|Q(D)| \geq \left(\frac{rmax(Q, D)}{rep(Q)}\right)^{C(chase(Q))}.$$

As with our proof of the upper bound in Theorem 4.7, we will iteratively transform $chase(Q)$ and the functional dependencies into a query Q' that has no functional dependencies such that $C(chase(Q)) = C(Q')$ and no step of the transformation increases the amount of potential increase in size of the query. Q' is obtained from $chase(Q)$ via the iterative application of the following substitution, which is equivalent to that employed in the proof of Theorem 4.7 in the case that the functional dependencies correspond to simple keys:

- Given a functional dependency $X_1, \ldots, X_k \to Y$, we remove it, and replace every instance of Y by the variables $Y_{X_1}, \ldots, Y_{X_k}$ in both the query and the functional dependencies, and add the variable Y_{X_i} to every atom that contains variable X_i.

While it was obvious that we would only require a small number of applications of the above substitutions in the case of simple keys, with arbitrary functional dependencies, it is not immediately obvious that the above process will terminate; after performing a substitution corresponding to the dependency $X_1, \ldots, X_k \to Y$, we might have increased the number of substitutions that we could make by a factor of k, and also increased the size of the left-hand sides of the substitutions by a factor of k. Nevertheless, the following lemma holds. We defer the proof to the extended version.

Lemma 6.4. *Consider s initial functional dependencies, whose left-hand sides have at most k variables each. After at most $\sum_{i=1}^{s} k^{2^{i-1}-1}$ substitutions we end up with a query, Q', that has no remaining functional dependencies.*

Given the above lemma, the proof of Proposition 6.3 now follows relatively easily, and we defer the details to the extended version.

Note that an analysis of the above process of translating instances corresponding to Q into instances corresponding to Q' also yields the following partial extension of Theorem 5.9:

Corollary 6.5. *Given a conjunctive query*

$$Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m),$$

and set of functional dependencies, $tw(Q(D))$ can not be bounded as any function of $tw(\langle R_1, \ldots, R_n \rangle)$, m, and u_i if there exists a valid coloring of $chase(Q)$ with 2 colors and color number 2.

While we cannot prove the general matching upper size bound for Proposition 6.3 as we had done in the case of simple keys, we can prove the following tight result for the case that $C(Q) = 1$ via pigeon-hole arguments.

Proposition 6.6. *A query $Q = R_0(u_0) \leftarrow R_1(u_1) \wedge \ldots \wedge R_n(u_m)$ with arbitrary functional dependencies is sparsity preserving if, and only if $C(chase(Q)) = 1$.*

Proof. If $C(Q) > 1$, the query is not sparsity preserving by Proposition 6.3. For the other direction, consider a query and set of functional dependencies for which there is an instance with $|Q(D)| > \text{rmax}(Q, D) \geq \max_{i \geq 1} |\pi_{u_i}(D)|$, where $\pi_{u_i}(D)$ denotes the projection of D onto the variables of u_i. Given such an instance, we will explicitly construct a valid coloring with color number greater than 1. First, we extend $Q(D)$ to $Q(D)'$ such that $|Q(D)| = |Q(D)'|$ and

$\pi_{u_0}(Q(D)') = Q(D)$, but each variable that appears in a u_i has a position in $Q(D)'$. (Such an extension trivially always exists, and need not be unique if $Q(D)$ is not the total join.)

Assume that the set of variables used in the u_i's is $X_1, \ldots, X_d$, and thus $d = arity(D')$. The construction of the coloring proceeds as follows: for each $i \geq 1$, by the pigeon-hole principle there must be some subset of the tuples, $Q(D)'_i$, with $|Q(D)'_i| = 2$ such that $|\pi_{u_i}(Q(D)'_i)| = 1$. For each variable X_j, we mark it with an i if $|\pi_{X_j}(Q(D)'_i)| = 2$. (Note that no $X_j \in u_i$ will be marked with an i.) After completing these markings for all $i \in \{1, \ldots, d\}$, we simply assign colors according to the markings: if variable X_j has not been marked, it is assigned the null color. If X_j has been marked with a single i, we assign it color i. If X_j has been marked with $i, \ldots, j$, we assign it the set of colors $(i, \ldots, j)$.

We now argue that the above coloring is valid. Consider a functional dependency $XY \to Z$. If the color i is in the label of Z, then $|\pi_{X_j}(Q(D)'_i)| = 2$, and thus in the two tuples of $Q(D)'_i$, position Z takes on two distinct values. The functional dependency implies that at least one of X or Y must also take on two distinct values, and thus at least one of X or Y will also be marked with an i, and therefore contain an i in its coloring, as desired. Finally, note that each of the m colors must be present in the label of some $X_j \in u_0$, because $|\pi_{u_0} Q(D)'| = |Q(D)|$, and thus there cannot be any u_j whose variables contain all m colors because as we noted above, no variable appearing in u_j can be marked with a j. Thus we have a valid coloring with m colors appearing in the final projection variables, with at most $m-1$ appearing together in any u_j, completing the proof. □

6.1 Open Question

The obvious remaining questions are whether the size bound of Proposition 6.3 is tight, and whether the condition that there is a coloring with 2 colors and color number 2 is a necessary condition for the treewidth to be unbounded. We believe that we are very close to proving that Proposition 6.3 is tight, via a direct approach that circumvents relying on Shearer's lemma for the upper bound.

On the complexity side, while Proposition 4.10 guarantees that computing the color number of a query with keyed relations can be done efficiently by expressing it as the solution to a succinct linear program, we do not currently know the complexity of computing the color number of a query with arbitrary functional dependencies. Especially in light of Proposition 6.6, even the complexity of deciding whether $C(Q) > 1$ remains a relevant open question, (though we note that it is $NP-complete$ to decide whether there is a coloring with 2 colors and $C(Q) > 1$).

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] A. V. Aho, C. Beeri, and J. D. Ullman. The theory of joins in relational databases. *ACM Trans. Database Syst.*, 4(3):297–314, 1979.

[3] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalence of relational expressions. *SIAM J. of Computing*, 8(2):218–246, May 1979.

[4] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[5] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *IEEE FOCS'08.*

[6] C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.

[7] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *ACM STOC*, 1977.

[8] S. Chaudhuri. An overview of query optimization in relational systems. In *PODS 1998.*

[9] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[10] A. Deutsch, L. Popa, and V. Tannen. Query reformulation with constraints. *SIGMOD Rec.*, 35(1):65–73, 2006.

[11] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, 2003.

[12] G. Gottlob and S. T. Lee. A logical approach to multicut problems. *Inf. Proc. Letters*, 103(4):136 – 141, 2007.

[13] G. Gottlob, S. T. Lee, and G. J. Valiant. Size and treewidth bounds for conjunctive queries (extended version). *Available from the authors upon request.*

[14] G. Gottlob, R. Pichler, and F. Wei. Efficient datalog abduction through bounded treewidth. In *AAAI*, pages 1626–1631, 2007.

[15] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA 2006.*

[16] P. J. Haas, J. F. Naughton, S. Seshadri, and A. N. Swami. Selectivity and cost estimation for joins based on random sampling. *J. Comput. Syst. Sci.*, 52(3):550–569, 1996.

[17] M. Jarke and J. Koch. Query optimization in database systems. *ACM Comput. Surv.*, 16(2):111–152, 1984.

[18] P. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS*, 2005.

[19] M. Lenzerini. Data integration: a theoretical perspective. In *PODS*, 2002.

[20] A. Y. Levy, A. O. Mendelzon, and Y. Sagiv. Answering queries using views. In *PODS 1995.*

[21] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[22] F. Olken and D. Rotem. Random sampling from database files: A survey. In *Proc. of Stat. and Scientific Database Management*, 1990.

[23] N. Robertson and P. D. Seymour. Graph minors II: Algorithmic Aspects of Tree-Width. *Journal of Algorithms*, 7:309–322, 1986.

[24] A. N. Swami and K. B. Schiefer. On the estimation of join result sizes. In *Advances in Database Technology - EDBT'94. 4th Int. Conf. on Extending Database Technology*, 1994.

[25] M. Thorup. Structured programs have small tree-width and good register allocation. *Information and Computation*, 142:318–332, 1998.