

# Conjunctive Query Evaluation by Search-Tree Revisited

Albert Atserias \*  
Universitat Politècnica de Catalunya  
Barcelona, Spain  
atserias@lsi.upc.edu

## Abstract

The most natural and perhaps most frequently used method for testing membership of an individual tuple in a conjunctive query is based on searching trees of partial solutions, or search-trees. We investigate the question of evaluating conjunctive queries with a time-bound guarantee that is measured as a function of the size of the optimal search-tree. We provide an algorithm that, given a database  $D$ , a conjunctive query  $Q$ , and a tuple  $a$ , tests whether  $Q(a)$  holds in  $D$  in time bounded by a polynomial in  $(sn)^{\log k} (sn)^{\log \log n}$  and  $n^r$ , where  $n$  is the size of the domain of the database,  $k$  is the number of bound variables of the conjunctive query,  $s$  is the size of the optimal search-tree, and  $r$  is the maximum arity of the relations. In many cases of interest, this bound is significantly smaller than the  $n^{O(k)}$  bound provided by the naive search-tree method. Moreover, our algorithm has the advantage of guaranteeing the bound for any given conjunctive query. In particular, it guarantees the bound for queries that admit an equivalent form that is much easier to evaluate, even when finding such a form is an NP-hard task. Concrete examples include the conjunctive queries that can be non-trivially folded into a conjunctive query of bounded size or bounded treewidth. All our results translate to the context of constraint-satisfaction problems via the well-publicized correspondence between both frameworks.

---

\*Partially supported by CICYT TIC2001-1577-C03-02

# 1 Introduction and Summary of Results

The foundational work of Chandra and Merlin [CM77] identified the class of conjunctive queries in relational database systems as an important and fundamental class of queries that are repeatedly “asked in practice”. These are the queries of first-order logic that are built from atomic formulas by means of conjunctions and existential quantification only. Thus, the generic conjunctive query takes the form

$$(\exists x_1) \cdots (\exists x_k)(R_1 \wedge \dots \wedge R_q)$$

where  $R_1, \dots, R_q$  are atomic formulas built from the relations of the database with the variables  $x_1, \dots, x_k$ . Conjunctive queries may also have free variables, but for the sake of simplicity we will focus on Boolean conjunctive queries in this introduction. Alternatively, it is known that the class of conjunctive queries coincides with the class of queries of the relational algebra that use selection, projection, and join only.

Evaluating conjunctive queries is such a common task that it is no surprise that a huge amount of work has focused on its algorithmic and complexity-theoretic aspects. The most obvious algorithm is perhaps the one that exhaustively checks for the existence of an assignment of values to the variables in such a way the relations in the body of the query (the quantifier-free part) are satisfied. Obviously, if the domain of the database has cardinality  $n$ , this algorithm takes time roughly  $n^k$ , which is exponential in the number of variables of the query. But, can we do better?

Unfortunately, unless  $P = NP$ , we cannot expect an algorithm that is polynomial in both  $n$  and  $k$  since the problem is NP-complete. This was already noticed by Chandra and Merlin [CM77]. To make things worse, more recent work on the parameterized complexity of query languages by Papadimitriou and Yannakakis [PY99] indicates that the situation might be even more dramatic. Namely, we cannot even expect an algorithm that, while arbitrarily complex in  $k$ , remains polynomial in  $n$ . Thus, we cannot expect an algorithm of complexity  $2^{2^k} n^2$ , say, unless certain widely believed assumptions in complexity theory are violated. These theoretical results indicate that the algorithmic problem is just too hard to be addressed in its wider generality.

Luckily, the situation in real database applications is not as catastrophic. Conjunctive queries that are asked in practice usually have some structure that makes them more tractable. The paradigmatic example is the class of *acyclic* conjunctive queries identified by Yannakakis [Yan81]. These are the conjunctive queries whose underlying hypergraph is acyclic, that is, the hypergraph that has the variables of the query as vertices, and the tuples of the variables appearing in the atomic formulas as hyperedges, is acyclic. Yannakakis showed that such queries could be evaluated in polynomial time by an efficient dynamic programming technique. The exact complexity of acyclic conjunctive queries was later studied in [GLS98], and generalized in several other directions [CR97, KV00]. The most interesting generalization is perhaps the one based on treewidth, to which we will get back later.

## 1.1 Search-trees and backtracking algorithms

Let us return now to the most obvious algorithm that checks for all possible assignments of values to the variables. Clearly, this algorithm can be modestly improved by a backtracking algorithm that considers the variables one-at-a-time and backtracks whenever the current partial assignment forces the body of the query to be either false because some atomic formula is falsified, or true because all atomic formulas are satisfied. Such a search-based algorithm can be remarkably fast in certain cases, especially if a good heuristic is used for choosing the next splitting variable. As a matter of fact, backtracking is probably the most frequently used method for solving constraint-satisfaction problems, which is essentially the same problem as conjunctive query evaluation as noticed by Kolaitis and Vardi [KV00], and is well-known by now.

This leads immediately to the concept of search-tree which is a key concept in our paper. A search-tree is an  $n$ -ary tree that is produced by such a backtracking procedure for an arbitrary choice of variables at

each branch. Here,  $n$  is the cardinality of the domain of the database. Notice that search-trees provide an enumeration of all possible solutions for the bound variables of the query since we backtrack even when the body of the query is satisfied. This permits us capturing the notion of optimal search-space through the concept of *minimal search-tree*. Intuitively, the size of the minimal search-tree for a given instance provides an ideal benchmark against which all search-based algorithms should be compared. For example, a backtracking algorithm that spends time  $O(n^k)$  on an instance admitting a search-tree of size  $O(kn)$  should be considered inefficient: it spends much more time than what is, in principle, necessary. Clearly, we would prefer an algorithm whose running time is bounded by a modest function of the size of the minimal search-tree. The ideal case would be an algorithm that is polynomial in that quantity.

The idea of comparing the efficiency of an algorithm with the size of the minimal search-tree originates in the field of propositional proof complexity, and, as far as we know, was not considered before in the fields of database theory and constraint-satisfaction problems. In proof complexity, the efficiency of a proof-search algorithm on a given propositional tautology is compared with respect to the size of its minimal proof in the proof system. A proof system admitting a proof-search algorithm that runs polynomially in the minimal proof is called *automatizable* [BPR00]. The connection shows up when the proof system under consideration is tree resolution and the instance is an unsatisfiable propositional formula  $F$  in conjunctive normal form. In that case, a minimal proof becomes a minimal search-tree for the constraint-satisfaction instance given by  $F$ , by simply turning it upside down (see also [BKPS02]).

## 1.2 Results of this paper

The main contribution of this paper is the observation that the concepts and techniques that were developed for automatizability of tree resolution carry over, to some extent, to the more general case of conjunctive query evaluation and constraint-satisfaction problems. By adapting an algorithm that was developed for tree resolution, we exhibit an algorithm for conjunctive query evaluation whose complexity is bounded by a non-trivial function of the size of the minimal search-tree.

More concretely, we provide an algorithm that, given a database  $\mathbf{A}$  of cardinality  $n$ , a tuple  $\mathbf{a}$  of  $\mathbf{A}$ , and a conjunctive query  $Q$  with  $k$  bound variables and relations of arity  $r$ , determines whether the Boolean conjunctive query  $Q(\mathbf{a})$  holds in  $\mathbf{A}$  in time that is polynomial in  $(sn)^{\log k} (sn)^{\log \log n}$  and  $n^r$ , where  $s$  is the size of the minimal search-tree for testing whether  $Q(\mathbf{a})$  holds in  $\mathbf{A}$ . While we do not achieve the desired polynomial bound on  $s$ , we note that the running time of our algorithm is remarkably good, compared to the obvious  $n^k$  bound, when the minimal search-tree is small. The algorithm is discussed in Section 3.

Then we go on to analyze our algorithm in Section 4. We first consider the class of conjunctive queries whose underlying graph is a tree, or is similar to a tree in the sense of having small treewidth. We note that if  $Q(\mathbf{a})$  has treewidth  $w$  and does not hold on  $\mathbf{A}$ , then the size of the minimal search tree is bounded by  $n^{(w+1)\log k}$ . Surprisingly perhaps, the hypothesis that  $Q(\mathbf{a})$  does not hold on  $\mathbf{A}$  seems essential for our proof. Nonetheless, this does not prevent us from showing that our algorithm works correctly for *any* query of bounded treewidth in time  $n^{O((\log k)^2)} n^{\log \log n}$ . Indeed, if the algorithm does not stop within the prescribed time bound, then we know that  $Q(\mathbf{a})$  holds in  $\mathbf{A}$ , although the algorithm gives no clue why.

It follows from this discussion that for queries of known treewidth  $w$ , our algorithm can be used for deciding whether  $Q(\mathbf{a})$  holds in  $\mathbf{A}$  within a time-bound that is far better than the worst case  $n^k$ , when  $k$  is large. Obviously, our bound is also far worse than the  $O(|Q|n^w)$  bound of the known ad-hoc algorithms for evaluating queries of treewidth  $w$  [GLS98, KV00]. It is quite interesting, nonetheless, that our algorithm achieves a non-trivial bound in that case despite it is not specialized for that purpose. As a matter of fact, our algorithm does not even compute a tree-decomposition of the query!

Another remarkable consequence is the following. In their seminal paper [CM77], Chandra and Merlin showed that for every conjunctive query there is a minimal equivalent query, unique up to isomorphism, that

can be obtained from the original one by identifying variables and deleting atomic formulas (see Theorem 12 and the discussion preceding it in [CM77]). In turn, Chandra and Merlin showed that finding such a minimal equivalent query is NP-hard. More recently, Dalmau, Kolaitis, and Vardi [DKV02] noticed that the problem remains NP-hard even when the minimal equivalent query is of constant size (and in particular has bounded treewidth). Thus, on the one hand, queries whose minimal equivalent query has bounded size admit search trees of size  $n^{O(1)}$  on databases on which they fail. The reason for this is that the minimal equivalent query is a subquery, so a search-tree for the minimal query is also a search-tree for the query itself, when the query evaluates to false. On the other hand, there is no efficient way of finding such a minimal equivalent query since the problem is NP-hard. Hence, it is perhaps surprising that, on those instances, our algorithm achieves complexity  $n^{O(\log k)} n^{\log \log n}$  without ever worrying about minimal equivalent queries at all. We elaborate further on this topic in Section 5.

Finally, in Section 6 we provide some lower bounds on the size of the minimal search-trees for certain conjunctive queries of interest. First, it is relatively easy to show that the minimal search-trees for the conjunctive query expressing the existence of a  $k$ -clique on graphs of size  $n$  may require  $n^{k-3}$  nodes. Second, it requires a slightly more complicated argument showing that the minimal search-trees for the conjunctive query expressing the existence of a path of length  $k$  on graphs of size  $n$  may require  $n^{\log k-3}$  nodes. This result shows that the  $n^{(w+1)\log k}$  upper bound for queries of treewidth  $w$  is essentially optimal. This is because the path-of-length- $k$  query has treewidth 1. Quite remarkably, our algorithm behaves in time polynomial in  $n^{(\log k)^2} n^{\log \log n}$  on such queries, which is nearly optimal with respect to search-tree size (for  $k$ 's larger than  $\log n$ ).

## 2 Preliminaries and Definitions

**Databases, structures, and conjunctive queries** We view databases as finite structures over finite relational vocabularies with constants. A *relational vocabulary with constants*  $\sigma$  is a set of *relation symbols*, each of a specified positive *arity*, and a set of *constant symbols*. A  $\sigma$ -*structure*, or *database*, consists of a *domain*  $A$ , a *relation*  $R^{\mathbf{A}} \subseteq A^r$  for each relation symbol  $R$  in  $\sigma$  of arity  $r$ , and an *individual*  $c^{\mathbf{A}} \in A$  for each constant symbol  $c$  in  $\sigma$ . Structures are denoted by

$$\mathbf{A} = (A, R_1^{\mathbf{A}}, \dots, R_t^{\mathbf{A}}, c_1^{\mathbf{A}}, \dots, c_d^{\mathbf{A}}),$$

where  $R_1, \dots, R_t$  are the relation symbols of  $\sigma$ , and  $c_1, \dots, c_d$  are the constant symbols of  $\sigma$ .

*Atomic formulas* are formulas of the form  $R(x_1, \dots, x_r)$  where  $R$  is a relation symbol of arity  $r$ , and  $x_1, \dots, x_r$  are first-order variables or constants. A *conjunctive query* is a formula of the form

$$(\exists z_1) \dots (\exists z_k) \psi,$$

where  $z_1, \dots, z_k$  are first-order variables, and  $\psi$  is a conjunction of atomic formulas. The quantifier-free part  $\psi$  is called the *body*. The variables  $z_1, \dots, z_l$  are called *bound variables*. The rest of variables of  $\psi$  are called *free variables*. The *total size* of a conjunctive query is the number of atomic formulas in  $\psi$ . Let  $Q$  be an atomic formula with free variables  $x_1, \dots, x_l$ . If  $\mathbf{A}$  is a  $\sigma$ -structure and  $\mathbf{a} = (a_1, \dots, a_l)$  is a tuple of  $\mathbf{A}$ , we write  $\mathbf{A} \models Q(\mathbf{a})$  if  $A$  satisfies  $Q(\mathbf{a})$  in the standard sense of first-order logic.

**Treewidth** Let  $\mathbf{G} = (V, E)$  be a finite graph. A *tree-decomposition* of  $\mathbf{G}$  is a pair  $(\{X_i : i \in I\}, T = (I, F))$  with  $\{X_i : i \in I\}$  a family of subsets of  $V$ , one for each node of  $T$ , and  $T$  is a tree such that:

1.  $\bigcup_{i \in I} X_i = V$
2. for all edges  $(v, w) \in E$ , there exists an  $i \in I$  with  $\{v, w\} \subseteq X_i$

3. for all  $i, j, k \in I$ : if  $j$  is on the path from  $i$  to  $k$  in  $T$ , then  $X_i \cap X_k \subseteq X_j$ .

The *width* of a tree-decomposition is  $\max_{i \in I} |X_i| - 1$ . The *treewidth* of  $\mathbf{G}$  is the minimum width over all possible tree-decompositions of  $\mathbf{G}$ .

The *treewidth of a  $\sigma$ -structure  $\mathbf{A}$*  is the treewidth of its *Gaifman graph*, that is, the graph whose set of vertices is  $A$ , and whose edges relate each pair of vertices that appear together in some tuple of the relations of  $\mathbf{A}$ . The *Gaifman graph of a conjunctive query  $Q$*  is the graph whose set of vertices is the set of bound variables of  $Q$ , and whose edges relate every pair of variables that appear together in an atomic formula (note that constants and free variables are ignored here). The *treewidth of a conjunctive query* is the treewidth of its Gaifman graph.

**Search-trees** Let  $\mathbf{A}$  be a finite  $\sigma$ -structure with universe  $A = \{a_1, \dots, a_n\}$ . Let  $f : V \rightarrow A$  be a partial mapping of the first-order variables to the universe  $A$  of  $\mathbf{A}$ . Extend  $f$  to the constant symbols of  $\sigma$  in the natural way. Let  $R(x_1, \dots, x_k)$  be an atomic formula. If  $x_i \in \text{Dom}(f)$  for every  $i \in \{1, \dots, k\}$ , we say that  $f$  *decides*  $R$ . If  $f$  decides  $R$  and  $(f(x_1), \dots, f(x_k)) \in R^{\mathbf{A}}$ , we say that  $f$  *satisfies*  $R$ . If  $f$  decides  $R$  and  $(f(x_1), \dots, f(x_k)) \notin R^{\mathbf{A}}$ , we say that  $f$  *falsifies*  $R$ . Let  $\psi(x_1, \dots, x_k)$  be a conjunction of atomic formulas. We say that  $f$  *satisfies*  $\psi$  if it satisfies every atomic formula in  $\psi$ . We say that  $f$  *falsifies*  $\psi$  if it falsifies some atomic formula in  $\psi$ . In those cases we say that  $f$  *decides*  $\psi$ . Otherwise, we say that  $f$  does not decide  $\psi$ .

A *search-tree* for  $\psi(x_1, \dots, x_k)$  in  $\mathbf{A}$  is a labeled rooted tree  $(T, L)$  whose nodes are labeled by partial assignments  $f : V \rightarrow A$ , and for which the following conditions are satisfied:

1. If  $v$  is the root of  $T$ , then  $L(v)$  is the empty partial assignment  $\emptyset$ .
2. If  $v$  is an internal node of  $T$ , then  $L(v)$  does not decide  $\psi$ .
3. If  $v$  is a leaf of  $T$ , then  $L(v)$  decides  $\psi$ .
4. If  $v$  is an internal node of  $T$  and  $L(v) = f$ , then there exists an  $x \notin \text{Dom}(f)$  such that  $v$  has exactly  $n$  successors  $v_1, \dots, v_n$  such that  $L(v_j) = f \cup \{(x, a_j)\}$  for every  $j \in \{1, \dots, n\}$ .

The variable  $x$  that is guaranteed to exist in clause 4 will be denoted by  $x(v)$ . We say that  $x(v)$  is the *splitting variable* at node  $v$ . Notice that there may be several search-trees for a given conjunction of atomic formulas and a given finite structure. A search-tree for  $\psi$  in  $\mathbf{A}$  is *minimal* if every other search-tree for  $\psi$  in  $\mathbf{A}$  is at least as large in size. For a finite  $\sigma$ -structure  $\mathbf{A}$ , a tuple  $\mathbf{a}$  of  $\mathbf{A}$ , and a conjunctive query  $Q$ , a *search-tree for testing whether  $\mathbf{A} \models Q(\mathbf{a})$*  is a search-tree for the body of  $Q(\mathbf{a})$ .

**Example 1** Let us illustrate the concept of search-tree. This example will also show how a single query can have multiple search-trees of very different sizes on a single database. Let us consider the vocabulary  $\sigma$  of one binary relation  $E$  and one unary relation  $P$ . Finite  $\sigma$ -structures in which  $E$  is symmetric and irreflexive are called *black-white colored graphs*. The tuples in  $E$  are called edges, and the points in  $P$  are called white vertices; the rest are called black vertices. Consider the query  $Q$  saying: “there exists a path of length  $k$  with a white end-point”. Formally,

$$(\exists x_1) \dots (\exists x_k) (E(x_1, x_2) \wedge E(x_2, x_3) \wedge \dots \wedge E(x_{k-1}, x_k) \wedge P(x_k)).$$

Let us consider now the black-white colored graph  $\mathbf{H}_{n,k}$  of Figure 1. It consists of a black  $n$ -clique followed by a path of length  $k$  attached to one of the vertices of the clique with all vertices black except the last that is white.

As a first example of a search-tree for testing whether  $\mathbf{H}_{n,k} \models Q$ , let us consider the one in which variables are queried in the order they appear in the query:  $x_1, x_2, \dots, x_k$ . The root is labeled by the empty

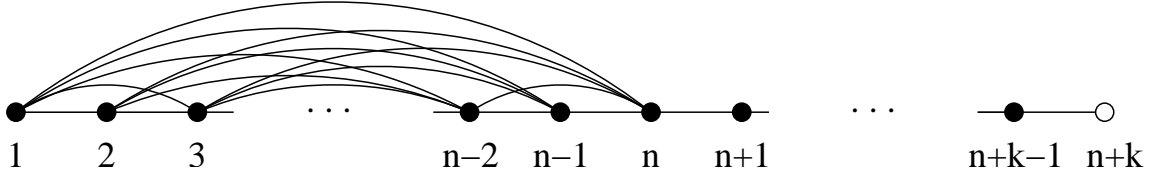


Figure 1:  $\mathbf{H}_{n,k}$

partial assignment, and has exactly  $n + k$  successors labeled by  $\{(x_1, 1)\}, \dots, \{(x_1, n + k)\}$ , respectively. Each such successor  $\{(x_1, i)\}$ , in turn, has exactly  $n + k$  successors: the  $j$ -th is labeled by  $\{(x_1, i), (x_2, j)\}$ . At this point, the vertex labeled by  $\{(x_1, n + k), (x_2, n + k)\}$  is declared a leaf because its partial assignment falsifies the body of the query (there is no edge between  $n + k$  and itself). A few other such nodes are declared leaves for the same reason. The rest of nodes have again  $n + k$  successors, and the tree goes on in this fashion until all variables are queried. Simple inspection reveals that the size of this tree is at least  $n^k$ . Let us consider now the tree that is built by querying the variables in reverse order,  $x_k, x_{k-1}, \dots, x_1$ . After  $x_k$  is queried, the only value for  $x_k$  that does not falsify the body is  $n + k$  (this is the only white vertex). Hence, all other vertices of the tree are declared leaves. Now, given the value  $n + k$  for  $x_k$ , there is again only one value for  $x_{k-1}$  that does not falsify the body and this is  $n + k - 1$  (the only vertex connected to  $n + k$ ). Following in this fashion we see that the size of this tree is bounded by  $k \cdot (n + k) + 1$ , which is qualitatively smaller than  $n^k$ . This shows that the choice of splitting variables can have a dramatic impact in the size of search-trees.  $\square$

In the particular case  $\mathbf{H}_{n,k} \models Q$  we gave in Example 1, it should be clear that the best strategy for a search-tree is to choose the variables in reverse order. In general, however, knowing which variable needs to be queried at each node to achieve optimal tree-size may be a difficult question. Let us point out that in the two examples we gave, the variables are all queried in the same order in each path of the search-tree. However, our definition of search-tree does not require that; indeed variables could be queried in arbitrary ways in different paths.

### 3 Booleanization and Algorithm

The purpose of this section is to develop the algorithm that achieves the promised performance. Let us start by announcing the result:

**Theorem 1** *Let  $\sigma$  be a relational vocabulary of maximum arity  $r$  and cardinality  $t$ . There exists a deterministic algorithm that, given a finite  $\sigma$ -structure  $\mathbf{A}$  of cardinality  $n$ , a conjunctive query  $Q$  with  $k$  bound variables and total size  $q$ , and a tuple  $\mathbf{a}$  from  $\mathbf{A}$ , determines whether  $\mathbf{A} \models Q(\mathbf{a})$  in time polynomial in  $q, t, n^r, k$ , and  $(sn)^{\log k} (sn)^{\log \log n}$ , where  $s$  is the size of a smallest search-tree for testing whether  $\mathbf{A} \models Q(\mathbf{a})$ .*

The proof of this theorem requires some preparation. The first thing we do is a *Booleanization* of the problem that will simplify the design and the analysis of the algorithm. Let  $A = \{a_1, \dots, a_n\}$  be the universe of  $\mathbf{A}$ . Each element of the universe  $a_i \in A$  can be encoded by a string of  $\log n$  bits that we denote by  $[a_i]$ . In turn, by using this encoding, each relation on  $A$  of arity  $r$  can be identified with a relation on the Boolean domain  $\{0, 1\}$  of arity  $r \log n$  as discussed next. For a finite  $\sigma$ -structure  $\mathbf{A}$ , we denote by  $\mathbf{A}^{(n)}$  the *Booleanization* of  $\mathbf{A}$ ; that is, the structure whose universe is  $\{0, 1\}$ , and that has one  $r \log n$ -ary relation  $R^{\mathbf{A}^{(n)}}$  for each  $r$ -ary relation symbol  $R$  in  $\sigma$ . The relation  $R^{\mathbf{A}^{(n)}}$  is defined as follows:

$$R^{\mathbf{A}^{(n)}} = \{[a_{i_1}] \cdots [a_{i_r}] : (a_{i_1}, \dots, a_{i_r}) \in R^{\mathbf{A}}\}.$$

For an  $r$ -tuple  $\mathbf{a}$ , let  $\mathbf{a}^{(n)}$  be the  $r \log n$ -tuple encoding  $\mathbf{a}$  over  $\{0, 1\}$ . Note that the Booleanization is not uniquely determined because the coding we chose depends on the particular ordering  $a_1, \dots, a_n$  of the elements of the universe. Note also that not every structure over  $\{0, 1\}$  is a Booleanization because some tuples may not code elements of  $A$ .

The Booleanization can also be carried out over a conjunctive query. If  $Q$  is a conjunctive query with  $k$  bound variables, its Booleanization  $Q^{(n)}$  is the conjunctive query with  $k \log n$  bound variables that results from using  $\log n$  new variables for each original variable in  $Q$ , and replacing the atomic formulas by their Booleanization.

**Example 2** Consider the vocabulary of Example 1 consisting of a single binary relation  $E$  and a single unary relation  $P$ . Let  $\mathbf{G} = (\{a, b, c, d, e\}, \{(a, b), (b, a), (c, d), (e, b)\}, \{c, e\})$  be a  $\sigma$ -structure. We need 3 bits to encode 5 elements: we code  $a$  through  $e$  in the order they appear as 000, 001, 010, 011, 100. The Booleanization of  $\mathbf{G}$  is thus:

$$\mathbf{G}^{(5)} = (\{0, 1\}, \{000001, 001000, 010011, 100001\}, \{010, 100\}).$$

Let  $Q$  be the query  $(\exists x_1)(\exists x_2)(E(x_1, x_2) \wedge P(x_2))$ . It's 5-th Booleanization is

$$\mathbf{Q}^{(5)} = (\exists x_1^1)(\exists x_1^2)(\exists x_1^3)(\exists x_2^1)(\exists x_2^2)(\exists x_2^3)(E(x_1^1, x_1^2, x_1^3, x_2^1, x_2^2, x_2^3) \wedge P(x_2^1, x_2^2, x_2^3)).$$

Note that, while the Booleanization of a structure depends on the particular order we chose for the elements of the universe, the Booleanization of a query is uniquely determined up to renaming of variables.  $\square$

The following Lemma is obvious.

**Lemma 1** *Let  $\mathbf{A}$  be a finite  $\sigma$ -structure of cardinality  $n$ , let  $\mathbf{a}$  be a tuple of  $\mathbf{A}$ , and let  $Q$  be a conjunctive query. Then  $\mathbf{A} \models Q(\mathbf{a})$  if and only if  $\mathbf{A}^{(n)} \models Q^{(n)}(\mathbf{a}^{(n)})$ . Moreover, if there exists a search tree for testing whether  $\mathbf{A} \models Q(\mathbf{a})$  of size  $s$ , then there exists a search tree for testing whether  $\mathbf{A}^{(n)} \models Q^{(n)}(\mathbf{a}^{(n)})$  of size  $2sn$ .*

*Proof:* Take the search tree for  $\mathbf{A} \models Q(\mathbf{a})$  and replace each internal node by a complete binary tree of height  $\log n$ . This blows up the tree by a factor of at most  $2n$ .  $\square$

The Booleanization allows us focus on the Boolean case, which is nothing else but a generalized satisfiability problem. Now we can apply the techniques that were developed for propositional logic and tree resolution [BP96, BKPS02].

Let  $\mathbf{A}$  be a Boolean  $\sigma$ -structure, that is, a  $\sigma$ -structure with Boolean domain  $A = \{0, 1\}$ . Let  $\mathbf{a}$  be a tuple of  $\mathbf{A}$ , and let  $Q$  be a conjunctive query. The algorithm takes a partial assignment  $f : V \rightarrow A$  as parameter and performs as follows: First, the algorithm checks whether  $f$  decides the body of  $Q(\mathbf{a})$ , in which case it returns the leaf-tree that consists of a single node labeled by  $f$ . Otherwise, for every variable  $x \notin \text{Dom}(f)$  and every value  $a \in \{0, 1\}$ , the algorithm calls recursively itself on input  $f \cup \{(x, a)\}$ . These recursive calls are run in parallel, either by executing one step from each in parallel rounds, or by applying a doubling technique that executes  $2^i$  steps of each call, sequentially, for increasing values of  $i$ . As soon as one of the recursive calls terminates, say, the one with input  $f \cup \{(x, a)\}$ , the rest of calls are aborted except for  $f \cup \{(x, 1 - a)\}$  which is run to completion. Let  $T_a$  and  $T_{1-a}$  be the search-trees returned by the only two recursive calls that are run to completion. The output is the search-tree  $(f, T_0, T_1)$ ; that is, the search-tree whose root is labeled by  $f$ , whose left subtree is  $T_0$ , and whose right subtree is  $T_1$ .

**Lemma 2** *Let  $\sigma$  be a relational vocabulary of maximum arity  $r$  and cardinality  $t$ . Let  $\mathbf{A}$  be a Boolean  $\sigma$ -structure, let  $\mathbf{a}$  be a tuple of  $\mathbf{A}$ , and let  $Q$  be a conjunctive query with  $k$  bound variables and total size  $q$ . The algorithm, when run with parameter  $f = \emptyset$ , returns a search-tree testing whether  $\mathbf{A} \models Q(\mathbf{a})$ . Moreover, if there exists such a search-tree of size  $s$ , then the algorithm runs in time polynomial in  $q, t, 2^r, k$  and  $s^{\log k}$ .*

*Proof:* The correctness of the algorithm is easily proved by induction on  $k$ . For the running time we proceed as follows. Let  $\psi$  be the body of  $Q$ . Let  $T(i, s)$  be the minimum upper bound to the running time of the algorithm for every  $f$  such that  $|\text{Dom}(f)| \geq k - i$  and the smallest search-tree for  $\psi[\mathbf{a}, f]$  has size at most  $s$ . When  $i = 0$ , the running time of the algorithm is bounded by some value  $c$  that depends on  $\sigma$  and  $Q$  only. More precisely, we can take  $c$  to be linear in  $qt2^r$ : for every atom in  $Q$ , we may need to test membership in a relation that has up to  $2^r$  tuples. Consider now the case  $i > 0$ . Consider a smallest search-tree of size at most  $s$ . If  $s \leq 1$ , the running time is again bounded by  $c$ , since necessarily,  $\emptyset$  decides  $\psi[\mathbf{a}, f]$ . If  $s \geq 2$ , one of its two subtrees has size at most  $s/2$ . It follows that at least one of the  $2i$  recursive calls terminates after at most  $T(i - 1, s/2)$  steps. Each parallel round takes  $di$  steps to execute for some constant  $d$ . The other recursive call that is left will take at most  $T(i - 1, s)$  steps to complete. All in all, the running time of the algorithm is bounded by

$$T(i, s) \leq c + diT(i - 1, s/2) + T(i - 1, s),$$

if  $i \geq 1$  and  $s \geq 2$ , and  $T(i, s) \leq c$  if either  $i = 0$  or  $s \leq 1$ . For solving this recurrence we expand the last term repeatedly, until we reach  $T(0, s) \leq c$ , and obtain

$$T(i, s) \leq c(i + 1) + d \sum_{j=1}^i jT(j - 1, s/2).$$

Now we use the fact that  $T(j, s/2) \leq T(j + 1, s/2)$  which follows directly from the definition of  $T$ , and obtain

$$T(i, s) \leq c(i + 1) + di^2T(i, s/2).$$

Solving this recurrence of a single variable  $s$  is now a routine task. If we replace  $\leq$  by  $=$  in the recurrence, the solution is

$$c \left[ (i + 1) \frac{(di^2)^{\log s + 1} - 1}{di^2 - 1} + (di^2)^{\log s} \right].$$

This is certainly an upper bound. Noting that  $(di^2)^{\log s} = s^{2 \log i + \log d}$  and recalling that  $c$  is linear in  $qt2^r$ , we see that the running time  $T(k, s)$  is bounded by a polynomial in  $q, t, 2^r, k$  and  $s^{\log k}$ .  $\square$

With this Lemma in hand we are ready to prove Theorem 1.

*Proof of Theorem 1:* It suffices to Booleanize  $\sigma$ ,  $\mathbf{A}$ ,  $Q$  and  $\mathbf{a}$ , and run the algorithm that we just described for the Boolean case. By Lemma 1, if  $\mathbf{A} \models Q(\mathbf{a})$  has a search-tree of size  $s$ , then  $\mathbf{A}^{(n)} \models Q^{(n)}(\mathbf{a}^{(n)})$  has a search-tree of size  $2sn$ . On the other hand, the number of bound variables of  $Q^{(n)}$  becomes  $k \log n$ , and the maximum arity of the Booleanization of  $\sigma$  becomes  $r \log n$ . The result follows by plugging these values into the bounds of Lemma 2.  $\square$

Let us note that, the way we described it, the algorithm does not produce a search-tree for  $\mathbf{A} \models Q(\mathbf{a})$ . This is because it is not necessarily possible to convert a search-tree for  $\mathbf{A}^{(n)} \models Q^{(n)}(\mathbf{a}^{(n)})$ , which is what the algorithm gives, into a search-tree for  $\mathbf{A} \models Q(\mathbf{a})$ , while preserving the bounds. Let us note, however, that a search-tree for  $\mathbf{A}^{(n)} \models Q^{(n)}(\mathbf{a}^{(n)})$  gives all the essential information. We do not know whether it is possible to have an algorithm with similar performance that avoids the Booleanization and produces a search-tree for  $\mathbf{A} \models Q(\mathbf{a})$ .

## 4 Search-Trees for Queries of Bounded Treewidth

The aim of this section is to investigate the size of search-trees for conjunctive queries whose underlying graph is a tree or is similar to a tree in the sense of having small treewidth. The key to the argument is that graphs of treewidth  $w$  have separators of size  $w + 1$ .

A  $p$ -separator of a graph  $\mathbf{G} = (V, E)$  is a set  $U \subseteq V$  such that each connected component of  $\mathbf{G} - U$  contains at most  $p$  vertices. The following fact is known about the relationship between treewidth and separator size (see [Bod98, Theorem 19]).

**Lemma 3** *Let  $\mathbf{G}$  be a graph of cardinality  $n$ . If the treewidth of  $\mathbf{G}$  is at most  $w$ , then  $\mathbf{G}$  has a  $\frac{1}{2}(n - w)$ -separator of size at most  $w + 1$ .*

We use this fact in the proof of the following Theorem. The proof of this result makes use of an idea that Moshe Vardi shared with the author.

**Theorem 2** *Let  $\sigma$  be a relational vocabulary of maximum arity  $r$  and cardinality  $t$ . Let  $\mathbf{A}$  be a finite  $\sigma$ -structure of cardinality  $n$ , let  $\mathbf{a}$  be a tuple of  $\mathbf{A}$ , and let  $Q$  be a conjunctive query with  $k$  bound variables. If  $Q(\mathbf{a})$  has treewidth at most  $w$  and  $\mathbf{A} \not\models Q(\mathbf{a})$ , then there exists a search-tree for testing whether  $\mathbf{A} \models Q(\mathbf{a})$  of size  $n^{(w+1) \log k}$ .*

*Proof:* We proceed by induction on  $k$ . If  $k = 0$  then the claim is obvious because the search-tree has size 1 (we convey here that  $\log 0 = 0$ ). Consider the case  $k > 0$ . Assume that  $Q(\mathbf{a})$  has treewidth at most  $w$  and  $\mathbf{A} \not\models Q(\mathbf{a})$ . Let  $\mathbf{G}$  be the Gaifman graph of  $Q(\mathbf{a})$ . Since  $\mathbf{G}$  has treewidth at most  $w$ , it has a  $\frac{1}{2}(k - w)$ -separator  $S = \{z_1, \dots, z_l\}$  of size at most  $w + 1$ . Let  $Q'(z_1, \dots, z_l)$  be the conjunctive query that results from  $Q(\mathbf{a})$  when the variables in  $S$  are left free. Since  $S$  is a  $\frac{1}{2}(k - w)$ -separator of  $\mathbf{G}$ , we may assume that  $Q'(z_1, \dots, z_l)$  is the conjunction of several conjunctive queries  $Q'_1(z_1, \dots, z_l), \dots, Q'_d(z_1, \dots, z_l)$  with at most  $\frac{1}{2}(k - w)$  bound variables each. Since  $\mathbf{A} \not\models Q(\mathbf{a})$ , we have  $\mathbf{A} \not\models Q'(f(z_1), \dots, f(z_l))$  for every partial assignment  $f$  for which  $\text{Dom}(f) = S$ . In turn, necessarily  $\mathbf{A} \not\models Q'_i(f(z_1), \dots, f(z_l))$  for some  $i \in \{1, \dots, d\}$ . Let  $i(f) \in \{1, \dots, d\}$  be such that  $\mathbf{A} \not\models Q'_{i(f)}(f(z_1), \dots, f(z_l))$ . Notice that the number of bound variables of  $Q'_{i(f)}$  is less than  $\frac{1}{2}k < k$ . We apply the induction hypothesis and obtain a search-tree for testing whether  $\mathbf{A} \models Q'_{i(f)}(f(z_1), \dots, f(z_l))$  of size  $n^{(w+1) \log(k/2)}$ . The search-tree for  $\mathbf{A} \models Q(\mathbf{a})$  can now be built by first querying the  $l \leq w + 1$  variables in the separator  $S$ , in sequence, and then, for each partial assignment  $f$  at the leaves of this partial search-tree, plugging in the search-tree for testing whether  $\mathbf{A} \models Q'_{i(f)}(f(z_1), \dots, f(z_l))$  that is given by the induction hypothesis. The size of the resulting tree is bounded by

$$n^{w+1} \cdot n^{(w+1) \log(k/2)} \leq n^{(w+1) \log k}$$

as was to be shown.  $\square$

In Section 6 we will show that the bound provided by Theorem 2 is essentially optimal even when the underlying graph of the query is a very simple tree. It is important to notice the extra hypothesis  $\mathbf{A} \not\models Q(\mathbf{a})$  in Theorem 2. As a matter of fact, we do not know if this hypothesis is necessary. In other words, we do not know if conjunctive queries of bounded treewidth always have search-trees of size  $n^{O(\log k)}$ .

## 5 Discussion: Bypassing the Core or Avoiding the Folding

Due to the extra hypothesis that the query does not hold in the database, the result in Theorem 2 does not give a complete algorithm for deciding if a conjunctive query of treewidth  $w$  holds on a given arbitrary database. However, if the treewidth is known beforehand, such an algorithm can be obtained nonetheless. Let us discuss this first.

## 5.1 Completing the algorithm through self-reducibility

Fix a relational vocabulary  $\sigma$  of maximum arity  $r$  and cardinality  $t$ . Suppose we run the algorithm of Section 3 on a  $\sigma$ -structure  $\mathbf{A}$  of cardinality  $n$  and a query  $Q(\mathbf{a})$  with  $k$  bound variables, total size  $q$ , and treewidth at most  $w$ . Let  $s = n^{(w+1) \log k}$ . By Theorem 1 and Theorem 2, we know that if  $\mathbf{A} \not\models Q(\mathbf{a})$ , then the algorithm finishes in a number of steps that is a fixed polynomial of  $q, t, n^r, k$ , and  $(sn)^{\log k} (sn)^{\log \log n}$ , and reports so. Consequently, if the algorithm does not succeed in finishing within that number of steps, we can conclude that  $\mathbf{A} \models Q(\mathbf{a})$ , although we get no clue why.

If we want to go around this difficulty, we can always use the self-reducibility of the problem to obtain a solution. Once we know that  $Q(\mathbf{a})$  holds on  $\mathbf{A}$ , we can cycle through all possible values for the first existentially quantified variable and run the algorithm again. One of these calls must detect that the remaining query is satisfiable. We fix that value for the first variable and then we go on to the next bound variable. Repeating this for every bound variable we end up with an assignment that satisfies the body of  $Q$ . Note that the running time of the procedure has increased only by a multiplicative factor of  $n \cdot k$ . Note as well that the approach works because the class of queries of treewidth at most  $w$  is closed under subqueries. In other words, if  $Q'$  is a query that is obtained from  $Q$  by removing an existential quantifier in  $Q$ , then the treewidth of  $Q'$  is at most that of  $Q$ .

It follows from this discussion that for queries of known treewidth, our algorithm can be used for deciding whether  $\mathbf{A} \models Q(\mathbf{a})$  within a time-bound that is far better than the worst case  $n^k$ , when  $k$  is large. Obviously, our bound is also far worse than the  $O(qn^w)$  bound of the known ad-hoc algorithms [] for evaluating queries of treewidth  $w$ . But there are two points we want to make. First, our algorithm is not special purpose for bounded treewidth queries and in fact does not even need a tree-decomposition of the query. It is not clear, however, if the saving in time by not computing the tree-decomposition compensates for the larger time bound. The second point is that the running time of the algorithm does not change even when the query itself does not have treewidth  $w$ , but its *core* has treewidth  $w$ . This point requires some explanation. In fact, the underlying idea is vastly more general, so let us devote a subsection to it.

## 5.2 Bypassing the core

Let us first introduce the key concepts of canonical databases and foldings of queries. Both these concepts are fundamental to conjunctive queries and go back to the original paper by Chandra and Merlin. For the rest of this section, we restrict the discussion to conjunctive queries without free variables.

Suppose  $Q = (\exists z_1) \dots (\exists z_k) \psi$  is a Boolean conjunctive query over  $\sigma$ , where  $\psi$  is a conjunction of atoms of the form  $R(z_{i_1}, \dots, z_{i_r})$ . The *canonical database* of  $Q$ , denoted by  $\mathbf{A}_Q$ , is the  $\sigma$ -structure whose universe is the set of variables  $\{z_1, \dots, z_k\}$ , and whose interpretation for the relation symbol  $R$  contains all tuples  $(z_{i_1}, \dots, z_{i_r})$  such that  $R(z_{i_1}, \dots, z_{i_r})$  is an atom in  $\psi$ . If  $\mathbf{A}_Q$  and  $\mathbf{A}_{Q'}$  are isomorphic, we say that the queries  $Q$  and  $Q'$  are also isomorphic. In simple terms, two queries are isomorphic if one is obtained from the other by renaming variables. Originally, the concept of the canonical database of a conjunctive query was introduced under the name of *natural model* by Chandra and Merlin.

Let us now remind the concept of homomorphism. Recall that a *homomorphism* between  $\sigma$ -structures  $\mathbf{A}$  and  $\mathbf{B}$  is a mapping  $h : A \rightarrow B$  such that, for every  $(a_1, \dots, a_r) \in R^{\mathbf{A}}$ , it holds that  $(h(a_1), \dots, h(a_r)) \in R^{\mathbf{B}}$ . If  $B \subseteq A$  and  $h$  is a homomorphism from  $\mathbf{A}$  to  $\mathbf{B}$  that fixes  $B$  pointwise, we say that  $h$  is a *retraction* from  $\mathbf{A}$  to  $\mathbf{B}$ . Using the concept of retraction, Chandra and Merlin introduced the concept of folding of a conjunctive query. We say that a conjunctive query  $Q'$  is a *folding* of another conjunctive query  $Q$  if the set of variables of  $Q'$  is a subset of the set of variables of  $Q$ , and there exists a retraction  $h : \mathbf{A}_Q \rightarrow \mathbf{A}_{Q'}$  such that  $\mathbf{A}_{Q'} = h(\mathbf{A}_Q)$ . Note that in that case,  $Q'$  is a subquery of  $Q$ . The first fundamental property about foldings is that the semantics of the query does not change:

**Lemma 4 ([CM77])** *Let  $Q$  be a Boolean conjunctive query, and let  $Q'$  be a folding of  $Q$ . Then  $Q$  and  $Q'$  are equivalent.*

By “equivalent” we mean, of course, that for every  $\sigma$ -structure  $\mathbf{A}$  we have that  $Q$  holds in  $\mathbf{A}$  if and only if  $Q'$  holds in  $\mathbf{A}$ . We write  $Q \equiv Q'$  when  $Q$  and  $Q'$  are equivalent. The second, and more important property about foldings is the fact that every conjunctive query has a unique minimal folding up to isomorphism:

**Theorem 3 ([CM77])** *Let  $Q$  be a Boolean conjunctive query. Then, there exists a folding  $Q_0$  of  $Q$  such that every conjunctive query  $Q'$  equivalent to  $Q$  has a folding  $Q'_0$  isomorphic to  $Q_0$ .*

It follows from the statement of this theorem, that the folding  $Q_0$  is minimal in the sense that no other folding can have less variables. Indeed, if  $Q'$  is an arbitrary folding of  $Q$ , then  $Q' \equiv Q$  by the Lemma, so  $Q'$  is isomorphic to  $Q_0$  by the Theorem, so  $Q'$  does not have less variables than  $Q_0$ . As it turns out, the canonical database  $\mathbf{A}_{Q_0}$  of the minimal folding  $Q_0$  of  $Q$  is exactly the *core* of  $\mathbf{A}_Q$ , that is, the unique minimal retract of  $\mathbf{A}$ . The concept of the core of a relational structure originated in graph theory (see for example [HN92]) and has played an important role in database theory in recent years [FKP05].

We are now in a position to discuss the role of minimal foldings in the complexity of evaluating conjunctive queries. Originally, minimal foldings were introduced for query optimization: since the minimal folding of  $Q$  does not depend on the database, it may be a good idea to compute the minimal folding once and for all and use it as an optimal optimization of  $Q$ . Unfortunately, finding the minimal folding is, in general NP-complete:

**Theorem 4 ([CM77])** *There exists a fixed conjunctive query  $P$  with three variables that is its own minimal folding and such that the following problem is NP-complete: “Given a Boolean conjunctive  $Q$ , is  $P$  the minimal folding of  $Q$ ?”*

For the interested reader, the proof consists of a simple reduction from the problem of 3-coloring a graph. This complexity results kills the idea of designing an efficient algorithm for conjunctive query evaluation by first computing the minimal folding of the query, and then evaluating it on the given database. It is for this reason that the following consequence to Theorem 1 may come as little surprise:

**Proposition 1** *Let  $\sigma$  be a relational vocabulary of maximum arity  $r$  and cardinality  $t$ . There exists a deterministic algorithm that, given a finite  $\sigma$ -structure  $\mathbf{A}$  of cardinality  $n$  and a conjunctive query  $Q$  with  $k$  bound variables and total size  $q$ , if  $\mathbf{A} \not\models Q$ , the algorithm returns a search-tree proving this in time polynomial in  $q, t, n^r, k$ , and  $n^{k^* \log k} n^{k^* \log \log n}$ , where  $k^*$  is the number of bound variables of the minimal folding of  $Q$ .*

Before we prove this proposition, let us note that the statement does not give any time guarantee when  $\mathbf{A} \models Q$ . As it turns out, in this case we cannot use the self-reducibility trick we described in Section 5.1 because the minimal folding is not preserved by variable-substitutions.

*Proof of Proposition 1:* Let  $T(q, t, r, k, s, n)$  be the running time of the algorithm in Theorem 1, where  $s$  is the size of the minimal search-tree for testing whether  $\mathbf{A} \models Q$ . We describe the algorithm in two steps: (a) first we assume that  $k^*$  is known, and (b) then we describe how to get rid of this assumption. Step (a): Suppose we knew  $k^*$ . Consider the following algorithm:

Run the algorithm in Theorem 1 for  $T(q, t, r, k, n^{k^*}, n)$  steps; if the algorithm terminates within that number of steps and returns a search-tree witnessing that  $\mathbf{A} \not\models Q$ , we return that search-tree. Otherwise, we return “don’t know”.

Let us now argue that this algorithm does what we want, assuming  $k^*$  is correct. Suppose that  $\mathbf{A} \not\models Q$ . In that case,  $\mathbf{A} \not\models Q_0$ , where  $Q_0$  is the minimal folding of  $Q$ . Since  $Q_0$  is a subquery of  $Q$  and  $\mathbf{A} \not\models Q_0$ , it follows that the minimal search-tree for testing whether  $\mathbf{A} \models Q_0$  is also a search-tree for testing whether  $\mathbf{A} \models Q$ . Its size is at most  $n^{k^*}$ . Hence, the algorithm of Theorem 1 terminates in  $T(q, t, r, n^{k^*}, s, n)$  steps and returns such a search-tree. This shows that the algorithm is correct. Step (b): Let us now see how to handle the general case in which  $k^*$  is not known. The idea is to try all possible values of  $k^*$ , starting at  $k^* = 1$ , until a search-tree witnessing that  $\mathbf{A} \not\models Q$  is found, if any. If  $\mathbf{A} \not\models Q$ , by the analysis above, the running time of this new algorithm is

$$\sum_{i=1}^{k^*} T(q, t, r, k, n^i, n).$$

Since  $T(q, t, r, k, s, n)$  is a polynomial in  $q, t, n^r$ , and  $(sn)^{\log k} (sn)^{\log \log n}$ , it follows immediately that the running time of the algorithm is as claimed in the statement of theorem.  $\square$

For the sake of comparison, let us remark that if an oracle told us the minimal folding of  $Q$ , it would be possible to find a search-tree witnessing that  $\mathbf{A} \not\models Q$  in time polynomial in  $n^{k^*}$ . What is surprising about Proposition 1 is that the bound  $n^{k^*}$  appears in the picture even though the algorithm does not worry about minimal foldings.

## 6 Bounds on Search-Tree Size

In this section we prove lower bounds for the minimal search-trees for particular queries of interest. The first lower bound is relatively easy, but we include the proof as a warm-up for the second, which is more difficult. The second lower bound shows that the  $n^{(w+1)\log k}$  bound for queries of treewidth  $w$  in Theorem 2 is essentially optimal.

### 6.1 Lower bound for the general case

Consider the vocabulary of graphs  $\sigma = \{E\}$ , where  $E$  is a binary relation symbol. For  $k \geq 2$ , let  $\text{CLIQUE}_k$  be the conjunctive query expressing the existence of a  $k$ -clique. More specifically,  $\text{CLIQUE}_k$  is the following conjunctive query:

$$(\exists x_1) \cdots (\exists x_k) \left( \bigwedge_{i \neq j} E(x_i, x_j) \right).$$

We aim for a family of graphs  $\mathbf{H}_n$  for which the size of the minimal search-trees for testing whether  $\mathbf{H}_n \models \text{CLIQUE}_k$  is nearly as large as it can be.

The graph  $\mathbf{H}_n$  that we need is the complete  $(k-1)$ -partite graph with all color-classes of cardinality  $n$ . More precisely, the set of vertices of  $\mathbf{H}_n$  is

$$V_n = \{(i, u) : 1 \leq i \leq k-1, 1 \leq u \leq n\},$$

and the set of edges of  $\mathbf{H}_n$  is

$$E_n = \{((i, u), (j, v)) : 1 \leq i, j \leq k-1, 1 \leq u, v \leq n, i \neq j\}.$$

Each set of vertices of the form  $\{(i, u) : 1 \leq u \leq n\}$  is called a color-class. Clearly,  $\mathbf{H}_n$  does not contain any  $k$ -clique, so the query  $\text{CLIQUE}_k$  does not hold on  $\mathbf{H}_n$ . Note that  $\mathbf{H}_n$  has  $(k-1)n$  vertices in total, and  $\text{CLIQUE}_k$  has  $k$  bound variables. Hence, the obvious upper bound for any search-tree is  $(kn)^k$ . We see next that when  $n$  is much bigger than  $k$ , then this is essentially the best one can do. The proof is quite simple but we give it as it will serve as a warm-up for a more difficult proof in the next section.

**Theorem 5** Every search-tree for testing whether  $\mathbf{H}_n \models \text{CLIQUE}_k$  has at least  $n^{k-3}$  nodes.

*Proof:* The idea of the proof is to describe an adversary argument that, given a purported search-tree of size less than  $n^{k-3}$ , finds a leaf that is labeled by a partial assignment that does not decide the body of  $\text{CLIQUE}_k$ . Since this contradicts the definition of search-tree, no such search-tree can exist.

Suppose that  $(T, L)$  is a search-tree testing whether  $\mathbf{H}_n \models \text{CLIQUE}_k$ . We construct a path  $q_0, q_1, \dots$  through  $T$ , starting at the root, with the following properties:

1.  $L(q_j)$  does not decide the body of  $\text{CLIQUE}_k$ .
2. The subtree rooted at  $q_j$  has size less than  $n^{k-3-j}$ .

The idea behind the construction is to set  $x(q_j)$  to a node of a different color-class; for example, we hope to set  $x(q_j)$  to a node in color-class  $j+1$ . Let  $q_0$  be the root of  $T$ . Suppose next that  $q_0, \dots, q_j$  have already been defined, and that  $q_j$  is not a leaf. We claim that among the  $n$  vertices in color-class  $j+1$ , there must exist at least one, say  $(j+1, u)$ , for which the subtree rooted of  $q_j$  labeled by  $L(q_j) \cup \{(x(q_j), (j+1, u))\}$  has size less than  $n^{k-3-j-1}$ . Indeed this is the case since otherwise the size of the subtree rooted at  $q_j$  would be at least  $n \cdot n^{k-3-j-1} = n^{k-3-j}$  which contradicts the inductive construction. Let  $q_{j+1}$  be any of these successors.

Notice that after a certain number of steps  $m$  no larger than  $k-3$ , we will reach a leaf  $q_m$  because the size of the subtree will become less than 2. It remains to be seen that our construction guarantees that the label  $L(q_m)$  of this leaf does not decide the body of  $\text{CLIQUE}_k$ . However, this is clear from the construction because the partial assignment that is built assigns each variable to a different color-class. Therefore,  $L(q_m)$  does not falsify any atomic formula, and it cannot satisfy all either because its domain is not all  $\{x_1, \dots, x_k\}$ . Hence,  $L(q_m)$  does not decide the body of  $\text{CLIQUE}_k$  as was to be shown.  $\square$

## 6.2 Lower bound for the bounded treewidth case

Consider now directed graphs. Again, we view them as structures over the vocabulary  $\sigma = \{E\}$  of one binary relation symbol  $E$ . For  $k \geq 2$ , let  $\text{PATH}_k(x, y)$  be the conjunctive query expressing the existence of a path of length  $k$  from  $x$  to  $y$ . More specifically,  $\text{PATH}_k(x, y)$  is the following conjunctive query:

$$(\exists x_1) \cdots (\exists x_{k-1})(E(x, x_1) \wedge E(x_1, x_2) \wedge \dots \wedge E(x_{k-2}, x_{k-1}) \wedge E(x_{k-1}, y)).$$

It is trivially seen that the treewidth of  $\text{PATH}_k(x, y)$  is one because the underlying Gaifman graph is a path, and hence a tree. We aim for a family of directed graphs  $\mathbf{G}_{k,n}$ , with two distinguished nodes  $s$  and  $t$ , for which the size of the minimal search-trees for testing whether  $\mathbf{G}_{k,n} \models \text{PATH}_k(s, t)$  nearly matches the upper bound provided by Theorem 2. Moreover, we will choose our graphs so that the hypothesis  $\mathbf{G}_{k,n} \not\models \text{PATH}_k(s, t)$  in that theorem is satisfied.

The construction of the directed graphs  $\mathbf{G}_{k,n}$  is illustrated in Figure 2. The set of vertices of  $\mathbf{G}_{k,n}$  is

$$V_{k,n} = \{(i, u) : 1 \leq i \leq k-1, 1 \leq u \leq n\} \cup \{s, t\}.$$

The vertices of the type  $(i, u)$  need to be thought as arranged into  $k-1$  columns of  $n$  vertices each. We call them middle vertices. The source vertex  $s$  is at column 0 and the target vertex  $t$  is at column  $k$ . Each middle vertex  $(i, u)$  at column  $i$  is connected precisely to the vertices at column  $i+1$  whose second components have the same parity as  $u$ . The source  $s$  is connected precisely to the vertices at column 1 whose second component is even, and the target  $t$  is connected precisely to the vertices at column  $k-1$  whose second component is odd. More formally, the arcs of  $\mathbf{G}_{k,n}$  are

$$\begin{aligned} E_{k,n} = & \{((i, u), (i+1, v)) : 1 \leq i \leq k-2, 1 \leq u, v \leq n, u \equiv v \pmod{2}\} \cup \\ & \{(s, (1, u)) : 1 \leq u \leq n, u \equiv 0 \pmod{2}\} \cup \\ & \{((k-1, u), t) : 1 \leq u \leq n, u \equiv 1 \pmod{2}\}. \end{aligned}$$

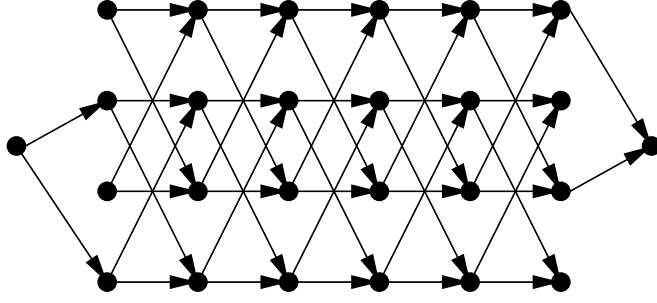


Figure 2:  $\mathbf{G}_{7,4}$

It is readily seen from the definition, that there is no path of length  $k$  from  $s$  to  $t$  in  $\mathbf{G}_{k,n}$ . In other words,  $\mathbf{G}_{k,n} \not\models \text{PATH}_k(s, t)$ . This is because the only middle vertices reachable from  $s$  are those whose second component is even, and the only middle vertices that reach  $t$  are those whose second component is odd.

**Theorem 6** For  $n \geq k/2 \geq 2$ , every search-tree for testing whether  $\mathbf{G}_{k,n} \models \text{PATH}_k(s, t)$  has at least  $n^{\log k - 3}$  nodes.

*Proof:* As in Theorem 5, the idea of the proof is again to describe an adversary argument. For simplicity we assume that  $n$  is an even number; the general case is similar. Suppose that  $(T, L)$  is a search-tree testing whether  $\mathbf{G}_{k,n} \models \text{PATH}_k(s, t)$ . Before we start the argument we need some terminology. Every internal node  $q$  of  $T$  has an associated column  $l(q)$  in  $\{1, \dots, k-1\}$  defined as follows. Let  $x(q) = x_i$ ; that is,  $x_i$  is the splitting variable at node  $q$ . Then we define  $l(q) = i$ .

We construct a path  $q_0, q_1, \dots$  through  $T$ , starting at the root, with the following properties:

1.  $L(q_j)$  does not decide the body of  $\text{PATH}_k(s, t)$ .
2. The subtree rooted at  $q_j$  has size less than  $2^j n^{\log k - 3 - j}$ .

Each internal  $q_j$  will also have an associated parity  $p_j \in \{0, 1\}$  that will be defined on the fly. Let  $q_0$  be the root of  $T$ . The parity  $p_0$  is defined 0 if  $2l(q_0) < k$  and 1 otherwise. Intuitively,  $p_0$  is 0 if column  $l(q_0)$  is closer to level 0 than to column  $k$ . Suppose next that  $q_0, \dots, q_j$  and  $p_0, \dots, p_j$  have already been defined, and that  $q_j$  is not a leaf. First we define the parity  $p_{j+1}$  as follows. Intuitively,  $p_{j+1}$  will be defined in such a way that the minimum distance, in terms of number of columns, between any two elements of different parity in the sequence is at most halved. More formally, consider the column  $l(q_j) = i$  of  $q_j$  and the column  $i'$  in  $\{l(q_0), \dots, l(q_{j-1}), 0, k\}$  that minimizes  $|i' - i|$  (break ties arbitrarily). If  $i' = 0$ , let  $p_{j+1} = 0$ . If  $i' = k$ , let  $p_{j+1} = 1$ . Otherwise, let  $j'$  be such that  $i' = l(q_{j'})$ , and let  $p_{j+1} = p_{j'}$ . Next we define  $q_{j+1}$ . We claim that among the  $n/2$  middle vertices at level  $i$  whose second component is congruent to  $p_{j+1} \pmod 2$ , there must exist at least one, say  $(i, u)$ , for which the subtree rooted at the successor of  $q_j$  labeled by  $L(q_j) \cup \{(x(q_j), (i, u))\}$  has size less than  $2^{j+1} n^{\log k - 3 - j - 1}$ . Indeed this is the case because otherwise the size of the subtree rooted at  $q_j$  would be at least

$$\frac{n}{2} \cdot 2^{j+1} n^{\log k - 3 - j - 1} = 2^j n^{\log k - 3 - j}$$

which contradicts the inductive construction. Let  $q_{j+1}$  be any of these successors.

Notice that after a certain number of steps  $m$  no larger than  $\log k - 2$ , we will reach a leaf  $q_m$  because the size of the subtree will become less than 2. It remains to be seen that our construction guarantees that the label  $L(q_m)$  of this leaf does not decide the body of  $\text{PATH}_k(s, t)$ . Consider the sequence  $q_0, \dots, q_m$ . To every internal  $q_j$  in the path there corresponds a vertex of  $\mathbf{G}_{k,n}$ , namely, the image of the variable  $x(q_j)$  under

$L(q_{j+1})$ . Let  $v_0, \dots, v_{m-1}$  be the corresponding sequence of vertices in  $\mathbf{G}_{k,n}$ . Note that, by construction, each  $v_j$  is a middle vertex of the form  $(l(q_j), u)$  and the parity  $p_j$  coincides with the parity of its second component  $u$ . Let us define  $v_m = s$ ,  $v_{m+1} = t$ ,  $p_m = 0$ , and  $p_{m+1} = 1$ . We claim that any two vertices in  $\{v_0, \dots, v_{m+1}\}$  that belong to consecutive levels are connected by an arc. In order to see this, it suffices to note that the shortest distance between any pair of elements of different parity in the sequence is at least  $k/2$  when  $j = 0$ , and is at most halved when going from  $j$  to  $j + 1$ . Therefore, by  $j = \log k - 2$ , the shortest distance between any pair of elements of different parity is at least 2. Hence, any two consecutive vertices have the same parity, so are connected by an arc. Hence,  $L(q_m)$  does not falsify any atomic formula, and it cannot satisfy all either because its domain is not all  $\{x_1, \dots, x_{k-1}\}$ . Hence,  $L(q_m)$  does not decide the body of  $\text{PATH}_k(s, t)$  as was to be shown.  $\square$

## 7 Conclusions

We have proposed a new way of measuring the complexity of algorithms for conjunctive query evaluation, or equivalently, for constraint-satisfaction problems. The concept of minimal search-tree wants to capture the notion of optimal search-space for search-based algorithms. As discussed in the introduction, measuring the complexity of the algorithm as a function of the minimal search-tree is an idea that originates in propositional proof complexity. By adapting an automatization algorithm for tree resolution that was developed in that context, we were able to provide an algorithm that achieves a remarkable theoretical performance. What remains to be seen is whether the idea can lead to practical algorithms with reasonable behavior.

Our work also suggests several technical open problems. First, our algorithm provides a search-tree for the Booleanization, but as we discussed, it is not clear that such a search-tree can be converted to a search-tree for the original conjunctive query. It would be nice to investigate this further. Second, proving the bounds on search-tree size for bounded treewidth queries seemed to require the hypothesis  $\mathbf{A} \not\equiv Q(\mathbf{a})$ . We do not know whether it is really needed. Let us state this as an

**Open Problem** Find bounds on the maximum search-tree size of conjunctive queries of bounded treewidth on structures on which they hold. More concretely: Do conjunctive queries with  $k$  variables and bounded treewidth have search-trees of size  $n^{O(\log k)}$  on structures of cardinality  $n$  on which they hold? If not, repeat for bounded pathwidth.

Another interesting direction to follow, that looks related to this work, is to establish the precise relationship between the CSP refutations developed in [AKV04] and the refutations provided by the search-trees when  $\mathbf{A} \not\equiv Q(\mathbf{a})$ . It seems that the techniques that were developed for proof complexity should be useful here. Ideally, it would be nice to move back and forth and apply techniques from one area to the other.

**Acknowledgments** I am grateful to José L. Balcázar and Roberto Nieuwenhuis for fruitful discussions, and also to a referee for comments. I am also grateful to Moshe Vardi for providing useful pointers and for the discussion of ideas related to Theorem 2.

## References

- [AKV04] A. Atserias, Ph. G. Kolaitis, and M. Vardi. Constraint propagation as a proof system. In *10th International Conference on Principles and Practice of Constraint Programming*, volume 3258 of *Lecture Notes in Computer Science*, pages 77–91. Springer-Verlag, 2004.
- [BKPS02] P. Beame, R. Karp, T. Pitassi, and M. Saks. The efficiency of resolution and Davis-Putnam procedures. *SIAM Journal of Computing*, pages 1048–1075, 2002.

- [Bod98] H. L. Bodlaender. A partial  $k$ -arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.
- [BP96] P. Beame and T. Pitassi. Simplified and improved resolution lower bounds. In *37th Annual IEEE Symposium on Foundations of Computer Science*, pages 274–282, 1996.
- [BPR00] M. L. Bonet, T. Pitassi, and R. Raz. On interpolation and automatization for Frege systems. *SIAM Journal of Computing*, 29(6):1939–1967, 2000. A preliminary version appeared in FOCS’97.
- [CM77] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *9th Annual ACM Symposium on the Theory of Computing*, pages 77–90, 1977.
- [CR97] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *6th International Conference on Database Theory*, volume 1997 of *Lecture Notes in Computer Science*, pages 56–70, 1997.
- [DKV02] V. Dalmau, Ph. G. Kolaitis, and M. Y. Vardi. Constraint satisfaction, bounded treewidth, and finite variable logics. In *8th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2470 of *Lecture Notes in Computer Science*, pages 310–326. Springer, 2002.
- [FKP05] R. Fagin, Ph. G. Kolaitis, and L. Popa. Data exchange: Getting to the core. *ACM Transactions on Database Theory*, 30(1):174–210, 2005.
- [GLS98] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. In *39th Annual IEEE Symposium on Foundations of Computer Science*, pages 706–715, 1998.
- [HN92] P. Hell and J. Nešetřil. The core of a graph. *Discrete Mathematics*, 109:117–126, 1992.
- [KV00] Ph. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2):302–332, 2000.
- [PY99] C. H. Papadimitriou and M. Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.
- [Yan81] M. Yannakakis. Algorithms for acyclic database schemes. In *7th International Conference on Very Large Data Bases*, pages 82–94, 1981.