A Theoretical and Experimental Comparison of Algorithms for the Containment of Conjunctive Queries with Negation

Khalil Ben Mohamed, Michel Leclère, and Marie-Laure Mugnier

University of Montpellier II, France, {benmohamed,leclere,mugnier}@lirmm.fr

Abstract. We tackle the containment problem for conjunctive queries with negation, which takes two queries q_1 and q_2 as input and asks if q_1 is contained in q_2. A general approach for solving this problem consists of considering all completions of q_1 (intuitively these completions represent all canonical databases that satisfy q_1) and checking if each completion yields the same answer on q_2. Since the total number of completions of q_1 is exponential in the size of q_1, several proposals have aimed at reducing the number (and the size) of the completions checked. In this paper, we propose a unifying framework for comparing algorithms following this general approach and define two kinds of heuristics for exploring the space of completions. Combining these heuristics with both classical kinds of traversals, i.e., depth-first and breadth-first, we obtain four algorithms that we compare experimentally with respect to running time on difficult instances of the containment problem.

1 Introduction

This paper is devoted to the containment problem for conjunctive queries with negation (denoted CQC$^\neg$ hereafter). Stated generally, the query containment problem takes two queries q_1 and q_2 as input, and asks if q_1 is contained in q_2 (noted $q_1 \sqsubseteq q_2$), i.e., if the set of answers to q_1 is included in the set of answers to q_2 for all databases (e.g. [AHV95]). It has long been recognized as a fundamental database problem, which underlies many tasks such as query evaluation and optimization [CM77][ASU79], rewriting queries using views [Hal01] or detecting independence of queries from database updates [LS93]. Positive conjunctive queries are considered as the basic database queries [CM77]. Conjunctive queries with negation extend this class with negation on subgoals.

When only positive conjunctive queries are considered, query containment checking is NP-complete [AHV95]. It can be solved by checking if there is a homomorphism from q_2 to q_1. When atomic negation is considered, the problem becomes much more complex: it is π_2^P-complete[1] [FNTU07][CM09]. A general approach for solving it, first presented in [LS93] for conjunctive queries with inequalities and adapted in [Ull97] for conjunctive queries with negation, consists in considering all ways of completing q_1 with positive information. Intuitively, this amounts to generating representatives of all

[1] π_2^P = *co*-(NP^{NP})

database instances that satisfy q_1. In [LM07], as in the present paper, negative information is also explicitly added. Such queries obtained from q_1 by adding missing information, either positively or negatively, are called completions of q_1 (and total completions if no more information can be added) hereafter. Then $q_1 \sqsubseteq q_2$ if and only if there is a homomorphism from q_2 to each total completion of q_1. However, the number of (total) completions of q_1 is exponential in the size of q_1. Several proposals have aimed at reducing the number and the size of completions ([WL03],[LM07],[BLM10a]). These proposals had not been compared yet, neither in depth from a theoretical side nor experimentally.

In this paper, we first define a unifying abstract framework, which allows us to express existing algorithms, hence to explain the principles underlying these algorithms, and to propose new algorithms. This framework relies on a space of completions of q_1 and the CQC$^\neg$ problem is reformulated as a search problem in this space. The family of algorithms we consider here perform a traversal of this space, thus build a search tree. But they essentially differ in the definition of the tree (i.e., how the children of a node are defined) and the strategy for generating the tree (i.e., in a depth-first or breadth-first way). Our second contribution consists in the experimental comparison of these algorithms. More precisely, our analysis yields four algorithm schemes, with both being close to existing proposals ([WL03],[LM07]); we implemented them, by making all algorithms benefiting from further specific heuristics experimentally evaluated in one of our former papers [BLM10a]. The experiments were made on random problem instances (i.e., pairs of conjunctive queries with negation) known to be difficult.

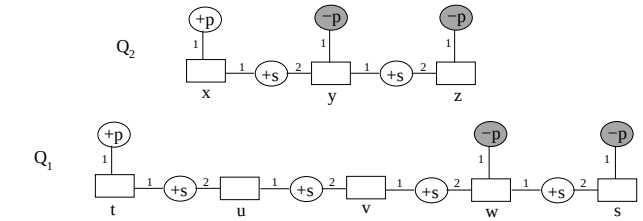
Paper layout. Section 2 is devoted to basic notions. Section 3 defines the abstract framework and containment checking methods within this framework. Section 4 presents algorithms and compare them experimentally. In section 5 the relationships of existing proposals with the evaluated algorithms are specified. Section 6 concludes this work and outlines perspectives.

2 Preliminaries

A *conjunctive query with negation ($CQ^\neg$)* is of the form: $q = ans(x_1, \ldots, x_q) \leftarrow p_1, \ldots, p_n, n_1, \ldots, n_m$, where each p_i (resp. n_i) is a positive (resp. negative) *subgoal*, $1 \leq n + m$, and *ans* is a special relation (which defines the answer part of the query). The left part of the query is called its *head* and the right part is its *body*. Each subgoal is of form $r(t_1, \ldots, t_k)$ (positive subgoal) or $\neg r(t_1, \ldots, t_k)$ (negative subgoal) where r is a relation and $t_1, \ldots, t_k$ is a tuple of terms (i.e., variables or constants). All variables $x_1, \ldots, x_q$ occur at least once in the body of the query. Without loss of generality, we assume that the same subgoal does not appear twice in the body of the query. A $CQ^\neg$ is *positive* if it has no negative subgoal ($m = 0$). A $CQ^\neg$ is *Boolean* if it has no variable in its head (we note *ans()*). In the following, we will focus on Boolean queries because having a non-empty *ans* part can only make the query containment problem easier.

As in [LM07], we will see $CQ^\neg$ as labeled graphs. More precisely, a $CQ^\neg$ q is represented as a bipartite, undirected and labeled graph Q, called *polarized graph (PG)*, with two kinds of nodes: term nodes and relation nodes. Each term of the query becomes a term node, that is unlabeled if it is a variable, otherwise it is labeled by the

constant itself. A positive (resp. negative) subgoal with relation r becomes a relation node labeled $+r$ (resp. $-r$) and it is linked to the nodes assigned to its terms. The labels on edges correspond to the position of each term in the subgoal (see Figure 1 for an example). For simplicity, the subgraph corresponding to a subgoal, i.e., induced by a relation node and its neighbors, is also called a *subgoal*. We note it $+r(t_1, \ldots, t_k)$ (resp. $-r(t_1, \ldots, t_k)$) if the relation node has label $+r$ (resp. $-r$) and list of neighbors $t_1, \ldots, t_k$. We note $\sim r(t_1, \ldots, t_k)$ a subgoal that can be positive or negative, i.e., $\sim \in \{+, -\}$. Subgoals $+r(t_1, \ldots, t_k)$ and $-r(u_1, \ldots, u_n)$ with the same relation but different signs are said to be *opposite*. Given a relation node label (resp. subgoal) l, $\overline{l}$ denotes the *complementary* relation node label (resp. subgoal) of l, i.e., it is obtained from l by reversing its sign. Queries are denoted by small letters (q_1 and q_2) and the associated graphs by the corresponding capital letters (Q_1 and Q_2). We note $Q_1 \sqsubseteq Q_2$ *iff* $q_1 \sqsubseteq q_2$. A PG is *consistent* if it does not contain two complementary subgoals.



$$q_1 = ans() \leftarrow p(t), s(t,u), s(u,v), s(v,w), \neg p(w), s(w,s), \neg p(s)$$
$$q_2 = ans() \leftarrow p(x), s(x,y), \neg p(y), s(y,z), \neg p(z)$$

Fig. 1. Polarized graphs associated with q_1 and q_2.

A *homomorphism* h from a PG Q_2 to a PG Q_1 is a mapping from term nodes of Q_2 to term nodes of Q_1 that satisfies: (1) if a term node is labeled by a constant, then its image has the same label (otherwise there is no constraint on the label of its image); (2) if $\sim r(t_1, \ldots, t_k)$ is a subgoal in Q_2 then $\sim r(h(t_1), \ldots, h(t_k))$ is a subgoal in Q_1. This notion can be seen as an extension to negative subgoals of the well-known query homomorphism (classically defined on positive queries). When there is a homomorphism h from Q_2 to Q_1, we say that Q_2 *maps* to Q_1 by h.

If Q_2 and Q_1 have only positive subgoals, $Q_1 \sqsubseteq Q_2$ *iff* Q_2 maps to Q_1. When they contain negative subgoals, only one side of this property remains true: if Q_2 maps to Q_1 then $Q_1 \sqsubseteq Q_2$; the converse is false, as shown in Example 1.

Example 1. See Figure 1: Q_2 does not map to Q_1 but $Q_1 \sqsubseteq Q_2$. Indeed, if we "complete" q_1 with all possible cases w.r.t. the relation p, we obtain the union of four queries
$q_{1,1} = ans() \leftarrow p(t), s(t,u), s(u,v), s(v,w), \neg p(w), s(w,s), \neg p(s), p(u), p(v)$,
$q_{1,2} = ans() \leftarrow p(t), s(t,u), s(u,v), s(v,w), \neg p(w), s(w,s), \neg p(s), \neg p(u), p(v)$,
$q_{1,3} = ans() \leftarrow p(t), s(t,u), s(u,v), s(v,w), \neg p(w), s(w,s), \neg p(s), p(u), \neg p(v)$ and
$q_{1,4} = ans() \leftarrow p(t), s(t,u), s(u,v), s(v,w), \neg p(w), s(w,s), \neg p(s), \neg p(u), \neg p(v)$.
Each of the queries corresponds to a possible way of completing q_1 w.r.t. p. Q_2 maps to each of the graphs associated with them. Thus q_1 is contained in q_2.

One way to solve CQC$^\neg$ is therefore to generate all "complete" PGs obtained from Q_1 using relations appearing in Q_1, and then to test if Q_2 maps to each of these graphs.

Definition 1 (Complete graph and completion). *Let Q be a consistent PG. It is* complete*, denoted Q^c, w.r.t. a set of relations $\mathcal{P}$, if for each $p \in \mathcal{P}$ with arity k, for each k-tuple of term nodes (not necessarily distinct) $t_1, \ldots, t_k$ in Q, it contains $+p(t_1, \ldots, t_k)$ or $-p(t_1, \ldots, t_k)$. A* completion *Q' of Q is a PG obtained from Q by repeatedly adding new relation nodes (on term nodes present in Q), without yielding inconsistency. Each addition is a* completion step*. A completion of Q is called* total *if it is a complete graph w.r.t. the set of relations considered, otherwise it is called* partial*.*

Theorem 1. *[LM07] Let Q_1 and Q_2 be two PGs (Q_1 consistent), $Q_1 \sqsubseteq Q_2$* iff *Q_2 maps to all total completions of Q_1 w.r.t. the set of relations appearing in Q_1.*

3 Methods for testing Containment

The complexity of a brute-force algorithm that would generate and test all completions of q_1 is prohibitive: $\mathcal{O}(2^{(n_{Q_1})^k \times |\mathcal{P}|} \times hom(Q_2, Q_1^c))$, where n_{Q_1} is the number of term nodes in Q_1, k is the maximum arity of a relation, $\mathcal{P}$ is the considered set of relations and $hom(Q_2, Q_1^c)$ is the complexity of checking the existence of a homomorphism[2] from Q_2 to Q_1^c.

Different tracks have been explored to reduce the number of homomorphisms performed. A first one is to reduce the number of considered total completions. [LM07] introduced the notion of *completion vocabulary* (denoted $\mathcal{V}$ in the following) which restricts the set of relations to consider for total completions: only relations appearing in opposite subgoals both in Q_2 and in Q_1 are to be considered. E.g. in Example 1 (Figure 1), $\mathcal{V} = \{p\}$.

A second studied track is to exploit partial completions. This idea, introduced in [WL03] and further developed in [LM07], aims at concluding about the containment before generating all total completions, by using some sufficient conditions on partial completions for success or failure of the containment.

3.1 The completion space

The completion space of a PG Q_1 (w.r.t. a given completion vocabulary) is the set of partial and total completions of Q_1 partially ordered by the relation "subgraph of". If Q_j is a descendant of Q_i, i.e., Q_i is a subgraph of Q_j, we say that Q_i *covers* Q_j. The *successors* of a completion are its immediate descendants (note that they only differ from it by one added subgoal).

Figure 2 shows the completion space of Q_1 from Example 1. Q_1 has four successors, namely the partial completions $Q_{1,1}$, $Q_{1,2}$, $Q_{1,3}$ and $Q_{1,4}$, obtained by adding respectively $+p(u)$, $+p(v)$, $-p(u)$ and $-p(v)$. From $Q_{1,1}$ we obtain two total completions $Q_{1,5}$ and $Q_{1,6}$ by adding respectively $+p(v)$ and $-p(v)$ (note that we cannot add $-p(u)$ because $+p(u)$ is present in $Q_{1,1}$), and similarly with the others $Q_{1,i}$. Finally, there are four total completions of Q_1.

[2] A brute-force algorithm for homomorphism check it in $\mathcal{O}(n_{Q_1}^{n_{Q_2}})$, where n_{Q_2} is the number of term nodes in Q_2.

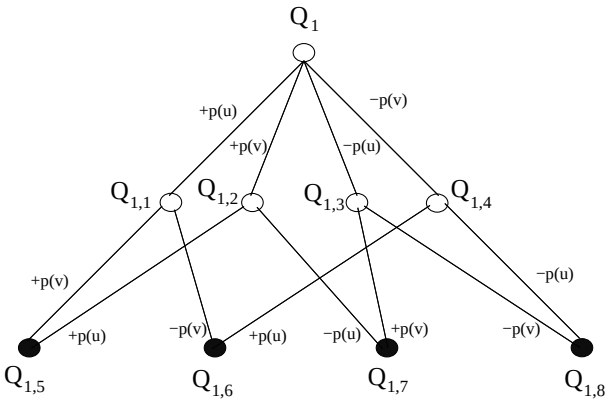


Fig. 2. The completion space of Example 1.

The following notion of a covering set is fundamental in this paper:

Definition 2 (Covering set). *Let Q_1 be a (consistent) query. A* covering set *of Q_1, noted $CS(Q_1) = \{Q_{1,1}, \ldots, Q_{1,n}\}$, is a set of (partial or total) completions of Q_1 such that every total completion Q_1^c of Q_1 is covered by a $Q_{1,i}$.*

Trivial examples of $CS(Q_1)$ are $\{Q_1\}$ and the set of all total completions of Q_1. The question "does $Q_1 \sqsubseteq Q_2$ hold ?" can now be recast as "is there $CS(Q_1) = \{Q_{1,1}, \ldots, Q_{1,n}\}$ such that Q_2 maps to each $Q_{1,i}$ for $i = 1 \ldots n$?"

The methods considered in this paper can be seen as exploring the completion space of Q_1, with the aim of finding a covering set of Q_1 such that Q_2 maps to each element of this set, or deciding that there is none. More precisely, they exploit the following property:

Property 1. Let Q_1 and Q_2 be two PGs, with Q_1 consistent. For all covering set $CS(Q_1)$, it holds that: $Q_1 \sqsubseteq Q_2$ if and only if, for each $Q_{1,i} \in CS(Q_1)$, $Q_{1,i} \sqsubseteq Q_2$.

Proof. If $Q_1 \sqsubseteq Q_2$ then for each partial or total completion Q'_1 of Q_1, $Q'_1 \sqsubseteq Q_2$; this is in particular true for elements of any $CS(Q_1)$. Conversely: let Q'_1 be a total completion of Q_1. By definition of a covering set, Q'_1 is covered by at least one $Q_{1,i} \in CS(Q_1)$, thus is a total completion of $Q_{1,i}$. Since $Q_{1,i} \sqsubseteq Q_2$, there is a homomorphism from Q_2 to any total completion of $Q_{1,i}$, in particular to Q'_1. □

This simple framework yields immediate proofs for the correctness of the algorithms studied in this paper.

3.2 Sufficient conditions for concluding

When exploring a current completion Q'_1, two kinds of sufficient conditions for concluding about containment or non-containment can be exploited, which are both based on homomorphism checks. Note that these conditions are not symmetrical.

Sufficient condition for concluding that $Q'_1 \sqsubseteq Q_2$ A simple sufficient condition for the containment of Q'_1 in Q_2 is the existence of a homomorphism from Q_2 to Q'_1 (cf. [LM07]). When this test is successful, it allows to "prune" the descendants of Q'_1: then there is necessarily a homomorphism from Q_2 to all graphs covered by Q'_1. Note however that it does not allow to conclude that Q_1 is contained in Q_2.

Sufficient conditions for concluding that $Q_1 \not\sqsubseteq Q_2$ Failure tests try to discover that there is at least one total completion Q_1^c covered by Q'_1 that does not admit any homomorphism from Q_2. These tests lead to a global negative answer, i.e., a negative answer about the initial containment problem. These failure tests exploit the property of some special subgraphs of Q_2, that must map by homomorphism to any completion of Q_1 (including Q_1), otherwise there exists a total completion Q_1^c to which Q_2 does not map. In the following, we call them "necessary subgraphs".

A first example of necessary graphs is given in [WL03]: the subgraph of Q_2 composed of all positive subgoals of Q_2. In [LM07], a more general characterization of such graphs is given: subgraphs without "exchangeable subgoals"; checking whether a graph is without exchangeable subgoals is NP-complete [MST09], but polynomially recognizable kinds of such graphs can be used, such as *pure subgraphs* (or independent subgraphs, which moreover exploit constraints induced by constants).

Definition 3 (pure subgraph). *A PG is said to be* pure *if it does not contain opposite subgoals (i.e., each relation appears only in one form, positive or negative). A* pure subgraph *of Q_2 is a subgraph of Q_2 that contains all term nodes in Q_2 (but not necessarily all relation nodes)*[3] *and is pure.*

See Figure 1: there are two pure subgraphs maximal for the inclusion: Q_2^+ contains $+p(x)$, $+s(x,y)$ and $+s(y,z)$; Q_2^- contains $-p(y)$, $-p(z)$, $+s(x,y)$ and $+s(y,z)$.

Moreover, the notion of necessary subgraphs goes with a more constrained homomorphism test, called *compatible homomorphism*. Intuitively, a homomorphism from a necessary subgraph of Q_2 to Q_1 is "compatible" if it can be extended to a homomorphism from Q_2 to a total completion of Q_1.

Definition 4 (Compatible homomorphism). *Let Q_2 and Q_1 be two PGs and Q'_2 be a necessary subgraph of Q_2. A homomorphism h from Q'_2 to Q_1 is said to be* compatible *w.r.t. Q_2 if, for each subgoal $\sim r(t_1,\ldots,t_k)$ in $Q_2 \setminus Q'_2$, the opposite subgoal $\overline{\sim r}(h(t_1),\ldots,h(t_k))$ is* not *in Q_1, and for each pair of opposite subgoals in $Q_2 \setminus Q'_2$, respectively on $(c_1,\ldots,c_k)$ and $(d_1,\ldots,d_k)$, $(h(c_1),\ldots,h(c_k)) \neq (h(d_1),\ldots,h(d_k))$.*

Property 2. [LM07] Let Q_1 and Q_2 be two PGs and Q'_2 be a necessary subgraph of Q_2. If there is no compatible homomorphism from Q'_2 to Q_1, then $Q_1 \not\sqsubseteq Q_2$.

3.3 Exploration heuristics of the completion space

The exploration of the completion space can be seen as an iterative procedure maintaining a covering set $CS(Q_1)$ and trying to find a "good" covering set, i.e., such that Q_2

[3] Note that this subgraph does not necessarily correspond to a set of subgoals because some term nodes may be isolated.

maps to each of its elements, or to show that there is none. Initially, $CS = \{Q_1\}$. At each step, the procedure performs the following:

1. pick a current completion Q'_1 in CS;
2. check if Q'_1 leads to conclude with a global failure;
3. otherwise: if Q_2 does not map to Q'_1, add to CS some successors of Q'_1, while keeping the property that CS is a covering set of Q_1.

When CS has been emptied, the global containment test succeeds. The set of built completions to which Q_2 maps can be seen as a proof that $Q_1 \sqsubseteq Q_2$.

Two ways of looking for a "good covering" set can be defined, which correspond to two exploration heuristics, called **dichotomic** and **"contradictAll"** hereafter. Note that, although not explicitly expressed as such, the proposals in [LM07] and in [WL03] can be seen as examples of these heuristics.

Before specifying them, let us consider the following notions:

Definition 5 (Missing subgoal, h-extension, h-contradiction). *Let Q_1 and Q_2 be two PGs (Q_1 consistent), Q'_2 a necessary subgraph of Q_2, and h a compatible homomorphism from Q'_2 to Q_1. Given $\sim r(t_1, \ldots, t_k)$ from $Q_2 \setminus Q'_2$, the subgoal $\sim r(h(t_1), \ldots, h(t_k))$ is said to be* missing *to Q_1 w.r.t. h if it is not in Q_1. A completion Q'_1 of Q_1 is called an* h-contradiction *if it contains the complementary of a missing subgoal w.r.t. h; otherwise it is called an* h-extension.

The dichotomic heuristic At each step, this heuristic partitions the completion space into two (disjoint) subspaces, by generating two completions from Q'_1 (the currently considered completion of Q_1): these completions are respectively obtained by adding a subgoal and its complementary. Since completions are consistent, it follows that the sets of completions covered by these newly generated completions are disjoint.

This method can be further specified by the choice of a necessary subgraph of Q_2, a compatible homomorphism h from this subgraph to Q'_1, and a missing subgoal to Q_1 w.r.t. h, so that the newly generated completions are respectively an h-extension and an h-contradiction of Q_1.

The correctness of this method is based on the following theorem:

Theorem 2. *Let Q_1 and Q_2 be two PGs. Then $Q_1 \sqsubseteq Q_2$ iff (1) there is a compatible homomorphism from a necessary subgraph of Q_2 [e.g. a pure subgraph] to Q_1 and (2) Let h be any such homomorphism. If there is a missing subgoal to Q_1 w.r.t. h, let $\sim r(t_1, \ldots, t_k)$ be such a subgoal; then $Q'_1 \sqsubseteq Q_2$ and $Q''_1 \sqsubseteq Q_2$, where Q'_1 (resp. Q''_1) is the h-contradiction (resp. h-extension) obtained from Q_1 by adding $\overline{\sim r}(t_1, \ldots, t_k)$ (resp. $\sim r(t_1, \ldots, t_k)$).*

Proof. Follows from Properties 1 and 2, and the fact that $\{Q'_1, Q''_1\}$ is a covering set of Q_1. □

Note that, if h is directly a homomorphism from Q_2 to Q_1, then there is no missing subgoal, and condition (2) is fulfilled.

The completion space is thus explored as a binary tree with Q_1 as root.

Figure 3 illustrates this method on Example 1, with Q_2^- as the necessary subgraph. Let $h_1 = \{x \mapsto v, y \mapsto w, z \mapsto s\}$ from Q_2^- to Q_1; $Q_{1,1}$ and $Q_{1,2}$ are built from Q_1, respectively by adding $+p(v)$ and $-p(v)$. Q_2 maps to $Q_{1,1}$, thus there is no need to complete $Q_{1,1}$. Q_2 does not map to $Q_{1,2}$: let $h_2 = \{x \mapsto u, y \mapsto v, z \mapsto w\}$ from Q_2^- to $Q_{1,2}$; $Q_{1,3}$ and $Q_{1,4}$ are built from $Q_{1,2}$, respectively by adding $+p(u)$ and $-p(u)$ to $Q_{1,2}$. Q_2 maps to $Q_{1,3}$ and to $Q_{1,4}$, respectively. Finally, the set proving that Q_1 is included in Q_2 is $\{Q_{1,1}, Q_{1,3}, Q_{1,4}\}$ (and there are four total completions of Q_1 w.r.t. p).

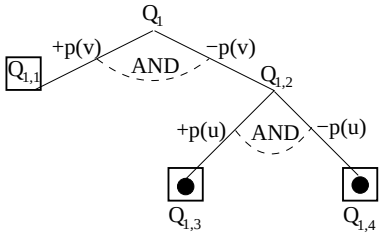


Fig. 3. A dichotomic search tree of Example 1. Each black dot represents a total completion and each square a partial one.

The "contradictAll" heuristic This heuristic is directly related to the notion of a compatible homomorphism from a necessary subgraph of Q_2 to Q'_1: at each step, it consists of choosing such a compatible homomorphism h to produce n h-contradictions, with each of them being obtained by adding to Q'_1 the complementary of one of the n missing subgoals to Q'_1 w.r.t. h.

The correctness of this method is based on the following theorem:

Theorem 3. *Let Q_1 and Q_2 be two PGs. Then $Q_1 \sqsubseteq Q_2$ iff (1) there is a compatible homomorphism from a necessary subgraph of Q_2 [e.g. a pure subgraph] to Q_1 and (2) Let h be any such homomorphism; then, for each missing subgoal $\sim r(t_1, \ldots, t_k)$ to Q_1 w.r.t. h, $Q_1^i \sqsubseteq Q_2$, where Q_1^i is the h-contradiction obtained from Q_1 by adding $\overline{\sim r}(t_1, \ldots, t_k)$.*

Proof. The covering of all total completions is ensured on the one hand by the construction of the n h-contradictions, and on the other hand by the h-extension $Q_1^{extension}$ (obtained from Q_1 by adding all the missing subgoals to Q_1 w.r.t. h) to which Q_2 maps. We conclude with Properties 1 and 2. □

Figure 4 illustrates this method on Example 1, with Q_2^+ as the necessary subgraph. Let $h_1 = \{x \mapsto t, y \mapsto u, z \mapsto v\}$ from Q_2^+ to Q_1; $Q_{1,1}$ and $Q_{1,2}$ are built from Q_1, respectively by adding $+p(v)$ and $+p(u)$. Note that Q_2 necessarily maps to $Q_1^{extension}$, obtained from Q_1 by adding $-p(v)$ and $-p(u)$. Q_2 maps to $Q_{1,1}$, thus there is no need to complete $Q_{1,1}$. Q_2 does not map to $Q_{1,2}$: let $h_2 = \{x \mapsto u, y \mapsto v, z \mapsto w\}$ from Q_2^+ to $Q_{1,2}$; $Q_{1,3}$ is built from $Q_{1,2}$ by adding $+p(u)$. As previously,

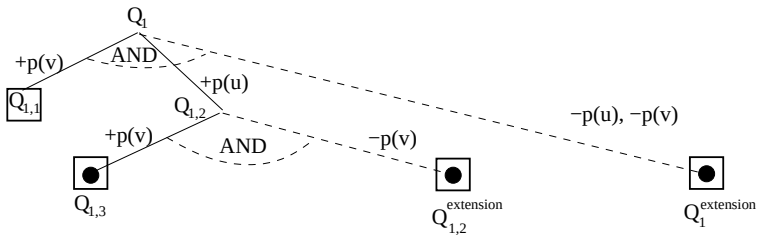


Fig. 4. A contradictAll search tree of Example 1.

Q_2 maps to $Q_{1_2}^{extension}$. Q_2 maps to $Q_{1,3}$. Finally, the set proving that $Q_1 \sqsubseteq Q_2$ is $\{Q_{1,1}, Q_{1,3}, Q_1^{extension}, Q_{1,2}^{extension}\}$.

Nevertheless, this space exploration, which does not partition the space, raises an important problem: it might be the case that the same completion is explored several times. In the worst case, this heuristic may lead to consider more completions than the brutal method that explores all total completions. To prevent these multiple explorations, two solutions can be imagined:

1. To forbid the construction of two identical completions. Then the algorithm becomes exponential in space because explored completions have to be memorized.
2. To "merge" identical completions at the end of each completion step (this works only with a breadth-first search algorithm, see Section 4.1). But the algorithm will be locally (i.e., at each completion step) exponential in space and this merging is expensive since it requires to pairwise compare the new completions.

In our experiments, we have compared the three alternatives: no prevention of multiple explorations, Solution 1 and Solution 2.

4 Comparison of Methods and Algorithms

In this section, we present several algorithms implementing the two heuristics presented in the previous section. Both heuristics perform traversals of the completion space, which differ in the way they select the successors of a node. Moreover, there are two classical ways of performing a traversal, namely in depth-first or in breadth-first way. We will first present two generic algorithms, corresponding to these two exploration schemes. Then, by concretizing the function that selects the successors of a node, we obtain the dichotomic or contradictAll heuristics. Thus, we finally obtain four algorithms, that we compare experimentally.

4.1 Breadth-first and Depth-first traversals

Algorithms 1 and 2 are generic algorithms that respectively perform a breadth-first and a depth-first search of the completion space. Algorithm 1 is iterative, it updates a covering set denoted by *CS*. We chose to present Algorithm 2 in a recursive way.

A negative answer to the test *if there is no homomorphism from* Q_2 allows to conclude that $Q_1 \sqsubseteq Q_2$ (cf. the sufficient condition in Section 3.2). It is an algorithmic optimization avoiding useless exploration of completions.

Subalgorithm dynamicFiltering(Q). This function corresponds to the sufficient condition for concluding that $Q_1 \not\sqsubseteq Q_2$ (cf. Section 3.2): if there is no compatible homomorphism from one (or several) necessary subgraph of Q_2 to Q then we can conclude that $Q_1 \not\sqsubseteq Q_2$.

Subalgorithm selectSuccessors(Q). This function returns a covering set of Q, which is a subset of the successors of Q. This subset depends on several variables:

1. a necessary subgraph, say Q'_2, that has to be mapped to Q;
2. a compatible homomorphism from Q'_2 to Q;
3. the chosen heuristic, i.e., dichotomic or contradictAll:
   (a) if we consider the dichotomic heuristic, we have also to choose a missing subgoal $\sim r(t_1, \ldots, t_k)$ to Q w.r.t. h. The function will then return the set $\{Q \cup \{\overline{\sim r}(t_1, \ldots, t_k)\}, Q \cup \{\sim r(t_1, \ldots, t_k)\}\}$; note that the order in which these two nodes are then explored is important, as shown in [BLM10a]: exploring first the h-contradiction (i.e., $\{Q \cup \overline{\sim r}(t_1, \ldots, t_k))$ is more efficient.
   (b) if we consider the contradictAll heuristic, the function will return the set $\{Q \cup \{\overline{\sim r}(t_1, \ldots, t_k)\}, \ldots, Q \cup \{\overline{\sim s}(u_1, \ldots, u_j)\}\}$ where $\sim r(t_1, \ldots, t_k), \ldots, \sim s(u_1, \ldots, u_j)$ are all the missing subgoals to Q w.r.t. h.

Algorithm 1: **breadthCheck**(Q_1, Q_2)

```
Input: two consistent PGs Q1 and Q2
Result: true if Q1 ⊑ Q2, false otherwise
begin
    CS ← {Q1};
    while CS ≠ ∅ do
        CS' ← ∅;
        foreach Q'1 ∈ CS do
            if there is no homomorphism from Q2 to Q'1 then
                if dynamicFiltering(Q'1) = failure then return false;
                else CS' ← CS' ∪ selectSuccessors(Q'1);
        CS ← CS';
    return true;
end
```

Finally, by combining dichotomic and contradictAll heuristics with breadth-first and depth-first searches, we obtain four algorithms: two breadth-first search ones, `dichotomicBreadthCheck` and `ContradictAllBreadthCheck`; two depth-first search ones, `dichotomicDepthCheck` and `ContradictAllDepthCheck`. In the next section we compare them experimentally.

Algorithm 2: **depthCheck**(Q_1, Q_2)

```
Input: two consistent PGs Q1 and Q2
Result: true if Q1 ⊑ Q2, false otherwise
begin
    if there is no homomorphism from Q2 to Q1 then
        if dynamicFiltering(Q1) = failure then return false;
        else
            foreach Q'1 ∈ selectSuccessors(Q1) do
                if depthCheck(Q'1,Q2) = false then return false;
    return true;
end
```

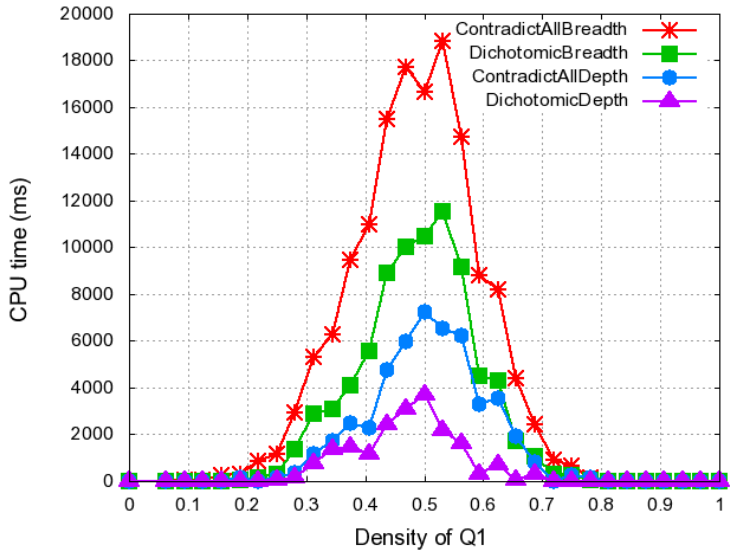
4.2 Experimental Comparison

We refer the reader to [BLM10a,BLM10b] for details about the experimental methodology. We built a random generator of polarized graphs and studied the influence of several parameters on the "difficulty" of problem instances (number of terms in the PG/query, percentage of constants, number of distinct relations, arity of these relations, density per relation, percentage of negation per relation). In the following experiments, we chose the parameter values shown to yield difficult instances. For each value of the varying parameter (density of Q_1 in the next experiments), we considered 500 instances and computed the mean search cost of the results on these instances. We also set a timeout at one minute[4].

We have made all four algorithms benefit from the improvements studied in [BLM10a]. Function **dynamicFiltering**(Q) performs filtering with all pure subgraphs of Q_2 maximal for the inclusion. Function **selectSuccessors**(Q) relies on a pure subgraph that is maximal for inclusion, say Q_2^{max}; the compatible homomorphism h from Q_2^{max} to Q is simply the first one found (as we have no criterion to choose among several such homomorphisms); in the case of dichotomic heuristic, the missing subgoal is randomly chosen among the set of missing subgoals to Q w.r.t. h. About avoiding multiple explorations, we compared experimentally the solutions proposed above and kept the best one for each algorithm: `ContradictAllBreadthCheck` uses a merging function and `ContradictAllDepthCheck` memorizes all explored completions.

Figure 5 shows the results obtained by the four algorithms on the same random CQC$^\neg$ instances. We can see that depth-first search algorithms (`dichotomicDepthCheck` and `ContradictAllDepthCheck`) are always better than breadth-first search ones. As expected, we can also see that dichotomic exploration is always better than contradictAll one (regardless of search strategy): this is due to the fact that dichotomic heuristic inherently avoids exploring twice the same completion, whereas contradictAll heuristic cannot ensure this property without a merging or memorizing function.

[4] With a timeout set at five minutes, breadth-first search algorithms lead to memory overflow.



Percentage of timeouts at a difficulty peak (with density of Q_1 = 0.5):
`CA-Breadth`=25%; `D-Breadth`=15%; `CA-Depth`=12%; `D-Depth`=4%.

Fig. 5. Comparison of the four algorithms.

5 Relationships with existing algorithms

In [LM07], Leclre and Mugnier proposed a depth-first search algorithm based on the dichotomic heuristic. We optimized this algorithm in [BLM10a], that led to a refined algorithm named `recCheckPlus`. This latter algorithm is exactly `dichotomicDepthCheck`.

In [WL03], Wei and Lausen proposed a breadth-first search algorithm (denoted by *WL-algorithm* hereafter) based on the following theorem, which we reformulate in our framework:

Theorem 4. *[WL03]. Let Q_1 and Q_2 be two PGs. Then, $Q_1 \sqsubseteq Q_2$ iff (1) there is a (compatible) homomorphism from Q_2^+ to Q_1 and (2) for each such homomorphism h and for each missing subgoal $\sim r(t_1, \ldots, t_k)$ to Q_1 w.r.t. h, $Q'_1 \sqsubseteq Q_2$, where Q'_1 is the h-contradiction obtained from Q_1 by adding $\overline{\sim r}(t_1, \ldots, t_k)$.*

Theorem 3 can be seen as a generalization of Theorem 4: at point (1), it considers a compatible homomorphism from any necessary subgraph of Q_2 to Q_1 (instead of Q_2^+), and at point (2), it avoids to test all (compatible) homomorphisms at each completion step (whereas Theorem 4 proof and *WL-algorithm* explicitly use this test). More precisely, Wei and Lausen proposed a space exploration where at each step, all homomorphisms from Q_2^+ to the current completion are to be considered. The search space

is then explored as a particular AND/OR tree[5], as shown on Figure 6. To prove that $Q_1 \sqsubseteq Q_2$, one has to prove that $((Q_{1,1} \sqsubseteq Q_2) \wedge \ldots \wedge (Q_{1,m} \sqsubseteq Q_2)) \vee \ldots \vee ((Q_{1,m+1} \sqsubseteq Q_2) \wedge \ldots \wedge (Q_{1,v} \sqsubseteq Q_2))$.

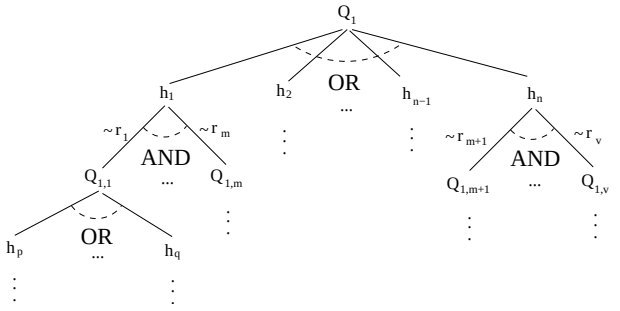


Fig. 6. "Wei and Lausen" exploration.

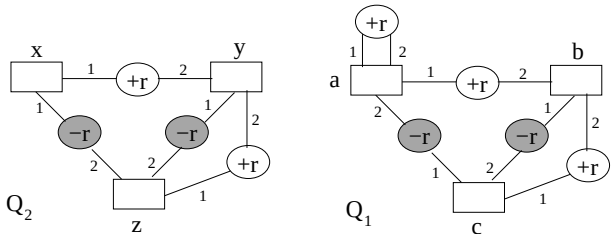
Let us comment on another aspect of *WL-algorithm*. This algorithm considers only "new" homomorphisms from Q_2^+ to Q_1' (the current completion), with the aim of avoiding multiple computation of the same homomorphisms. There are several ways of interpreting the notion of a "new" homomorphism:

1. it is new w.r.t. the path from Q_1 to Q_1', i.e., it has not been computed during the generation of this path;
2. it is new w.r.t. the subtree composed of the descendants of Q_1' and their brothers;
3. it is new w.r.t. the entire tree.

The first possibility is necessarily fulfilled, because all explored completions are h-contradictions. Indeed, added subgoals contradict homomorphisms used throughout the path from Q_1 to Q_1'. A new homomorphism according to the second definition is such that at least one subgoal in Q_2^+ is mapped to the subgoal added at the previous completion step. More precisely, let Q_1' be a completion obtained by adding $+r(e_1, \ldots, e_k)$. A homomorphism h from Q_2^+ to Q_1' is said *new* if there is $+r(t_1, \ldots, t_k)$ in Q_2^+ such that $+r(h(t_1), \ldots, h(t_k)) \in Q_1'$ and $h(t_1), \ldots, h(t_k) = e_1, \ldots, e_k$. However, this definition of a new homomorphism is unsatisfactory for two main reasons:

1. It does not avoid multiple explorations of the same completion. In Figure 6 for example, completions $Q_{1,m}$ and $Q_{1,m+1}$ could be obtained by adding the same subgoal.
2. It makes *WL-algorithm* incomplete, i.e., this algorithm can miss solutions. Queries of Figure 7 illustrate this problem: $Q_1 \sqsubseteq Q_2$ whereas *WL-algorithm* concludes that $Q_1 \not\sqsubseteq Q_2$, because at the second completion step it does not find any **new** homomorphism (but it would continue if it considered all homomorphisms).

[5] Strictly speaking, it is not exactly an AND/OR tree because one of the halting conditions is global (which is based on Property 2), i.e., it allows to completely stop the process.



$$q_1 = ans() \leftarrow r(a,a), r(a,b), r(c,b), \neg r(b,c), \neg r(c,a)$$
$$q_2 = ans() \leftarrow r(x,y), r(z,y), \neg r(x,z), \neg r(y,z)$$

Fig. 7. A counterexample to the "new homomorphism" property.

The last possibility makes *WL-algorithm* incomplete as well: the completion process stops whereas it should continue by "reusing" some homomorphisms of other paths (the previous counterexample works here too: at the second completion step, there are no "new" homomorphisms w.r.t. the entire tree).

In light of this analysis, it appears that homomorphism novelty is not a good notion for the needs of the algorithm. It has to consider the novelty of a completion.

In summary, both `contradictAllDepthCheck` and `contradictAllBreadthCheck` are implementations of (a generalization of) the theorem of [WL03]. Algorithm `contradictAllBreadthCheck` can be seen as a clean implementation of the algorithm proposed in the Appendix of [WL03]. Moreover, it is expressed in a very simple way, which allows to easily check its correctness. Nevertheless, algorithm `ContradictAllDepthCheck`, which is as simple to express, is better (cf. Figure 5).

Let us end this section by briefly mentioning another algorithm proposed in [BLM10a]. Its way of exploring the space is totally different: it builds a *candidate* covering set at once, and then check if this set is actually a covering set by transforming it into a propositional logical formula and checking its unsatisfiability. Then, $Q_1 \sqsubseteq Q_2$ if and only if this formula is unsatisfiable. Nevertheless, this algorithm has been experimentally shown less efficient than `dichotomicDepthCheck` on difficult instances.

6 Conclusion

In this paper, we propose a unifying framework for comparing algorithms solving CQC$^\neg$, and define two kinds of heuristics: dichotomic and contradictAll. Combining these heuristics with both classical kinds of traversals, i.e., depth-first and breadth-first, we obtain four algorithms. We compare them experimentally and show that the depth-first search algorithm with dichotomic heuristic (`dichotomicDepthCheck`) is more efficient than the three others.

Real-world queries expressed by a user generally contain constants. In our experiments, we considered queries without any constants because we focused on difficult instances. Moreover, we checked that CQC$^\neg$ difficulty decreases very quickly with the increasing of the percentage of constants. As for further work, we will study how con-

stants can be exploited in algorithms, with the aim of drastically increasing the size of queries that can be processed within reasonable time.

Another perspective would be to compare `dichotomicDepthCheck` with logical provers solving problems of the same complexity class (Π_2^p-complete), such as the problem *2-QBF* (e.g. [GW99]).

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley, 1995.

[ASU79] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM J. Comput.*, 8(2):218–246, 1979.

[BLM10a] K. Ben Mohamed, M. Leclère, and M.-L. Mugnier. Containment of conjunctive queries with negation: Algorithms and experiments. In *DEXA*, pages 330–345, aug 2010.

[BLM10b] K. Ben Mohamed, M. Leclère, and M.-L. Mugnier. Deduction in existential conjunctive first-order logic: an algorithm and experiments. Technical Report RR-10010, LIRMM, mar 2010.

[CM77] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *9th ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[CM09] M. Chein and M.-L. Mugnier. *Graph-based Knowledge Representation and Reasoning—Computational Foundations of Conceptual Graphs*. Advanced Information and Knowledge Processing. Springer, 2009.

[FNTU07] C. Farré, W. Nutt, E. Teniente, and T. Urpí. Containment of conjunctive queries over databases with null values. In *ICDT 2007*, volume 4353 of *LNCS*, pages 389–403. Springer, 2007.

[GW99] I. P. Gent and T. Walsh. Beyond np: the qsat phase transition. pages 648–653. AAAI Press, 1999.

[Hal01] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[LM07] M. Leclère and M.-L. Mugnier. Some Algorithmic Improvments for the Containment Problem of Conjunctive Queries with Negation. In *Proc. of ICDT'07*, volume 4353 of *LNCS*, pages 401–418. Springer, 2007.

[LS93] A. Y. Levy and Y. Sagiv. Queries independent of updates. In *VLDB '93: Proceedings of the 19th International Conference on Very Large Data Bases*, pages 171–181, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[MST09] M.-L. Mugnier, G. Simonet, and M. Thomazo. On the complexity of deduction in existential conjunctive first-order logic (long version). Technical Report RR-09026, LIRMM, sep 2009.

[Ull97] J. D. Ullman. Information Integration Using Logical Views. In *Proc. of ICDT'97*, volume 1186 of *LNCS*, pages 19–40. Springer, 1997.

[WL03] F. Wei and G. Lausen. Containment of Conjunctive Queries with Safe Negation. In *International Conference on Database Theory (ICDT)*, 2003.