Containment of Conjunctive Queries with Negation: Algorithms and Experiments

Khalil Ben Mohamed, Michel Leclère, Marie-Laure Mugnier

Containment of Conjunctive Queries with Negation: Algorithms and Experiments

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

LIRMM (CNRS - University of Montpellier), France,
{benmohamed,leclere,mugnier}@lirmm.fr

Abstract. We consider the containment problem for conjunctive queries with atomic negation. Firstly, we refine an existing algorithm based on homomorphism checks, which itself improves other known algorithms in databases, and analyze it experimentally. Secondly, we present a new algorithm based on the translation of the containment problem into the problem of checking the unsatisfiability of a propositional logical formula, which allows us to use a SAT solver, and we experimentally compare both algorithms.

1 Introduction

The query containment problem is a fundamental problem in databases. It 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]). Algorithms based on query containment can be used to solve various problems, such as query evaluation and optimization [CM77][ASU79], rewriting queries using views [Hal01], detecting independence of queries from database updates [LS93], etc. In this paper, we consider the problem of deciding on containment for conjunctive queries with atomic negation (denoted CQC$^\neg$ hereafter). The so-called (positive) conjunctive queries form a class of natural and frequently used queries and are considered as the basic database queries [CM77]. Conjunctive queries with negation extend this class with negation on atoms. Note that CQC$^\neg$ is equivalent to important problems in artificial intelligence, such that: checking entailment / deduction between two first-order logic clauses (without function); query answering with boolean conjunctive queries with negation on a knowledge base composed of a set of positive and negative factual assertions (while making the open-world assumption).

When only positive conjunctive queries are considered, query containment checking is NP-complete [AHV95]. When atomic negation is considered, the problem becomes much more complex: it is π_2^P-complete[1] [FNTU07][CM09] and very few algorithms for solving it can be found in the literature.

This paper is devoted to refining and proposing algorithms solving CQC$^\neg$ and testing them experimentally. An algorithm scheme was introduced in [LM07], which itself improves the previous proposals in [Ull97] and [WL03]. All three algorithms use homomorphism as a core notion. We first compare experimentally several heuristics, which

[1] $\pi_2^P = (co\text{-}NP)^{NP}$

allows to refine this algorithm scheme. We then propose another approach, which consists of building a propositional logical formula from the queries q_1 and q_2, such that q_1 is contained in q_2 if and only if this formula is valid, i.e. always true. Equivalently, the negation of this formula is unsatisfiable, which allows to use a SAT solver (SAT is the problem of deciding whether a given propositional formula is satisfiable) and benefit from practical improvements achieved in this domain [Sai08]. However, the translation of the queries into the propositional formula is generally exponential in the size of the queries. Thus the question is whether –or when– the second algorithm can be better than the first one. We provide first experimental answers to this question.

Due to the lack of benchmarks or real-world data available for CQC$^\neg$, we built a random generator. We analyzed the influence of several parameter values on the problem instance difficulty in order to define difficult instances, on which the algorithms were run. In databases, conjunctive queries with negation are generally imposed to be *safe*, i.e. all variables in the query must occur in at least one positive subgoal. Hence, even if all of our results hold for general conjunctive queries with negation, we restrict the experiments to safe queries in this paper.

Paper layout. Section 2 recalls the framework of [LM07] and expresses previously proposed algorithm schemes in this framework. In Section 3, we present our experimental methodology and use it to compare several heuristics, which leads to propose a refined algorithm. The second approach is presented and compared to the refined algorithm in 4. Section 5 outlines the prospects of this work.

2 Framework

We recall here some basic definitions about queries with Datalog-like notations. More details can be found in [AHV95] for instance. A *conjunctive query with negation ($CQ^\neg$)* is of the form: $q = ans(x_1 \dots x_q) \leftarrow p_1, \dots, p_n, n_1, \dots, n_m$, 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 $r(t_1, \dots, t_k)$ (positive subgoal) or $\neg r(t_1, \dots, t_k)$ (negative subgoal) where r is a relation and $t_1, \dots, t_k$ is a tuple of terms (i.e. variables or constants). All variables $x_1 \dots x_q$ 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 *boolean* if it has no variable in its head (we note *ans()*). A $CQ^\neg$ is *positive* if it has no negative subgoal ($m = 0$). A $CQ^\neg$ is *safe* if each variable occurring in a negative subgoal also occurs in a positive one.

In the following, we will focus on boolean queries because having a non-empty *ans* part can only make the query containment problem easier. For the same reason, we can consider that queries contain no constants. Note however that the framework and all results hold for general $CQ^\neg$.

In [LM07], $CQ^\neg$ are seen as labeled graphs. This allows to rely on graph notions that have no simple equivalent in logic (such as pure subgraphs, see later). 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*. Opposite Subgoals $+r(t_1, \ldots, t_k)$ and $-r(t_1, \ldots, t_k)$ with the same list of arguments are said to be *contradictory*. 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 contradictory subgoals.

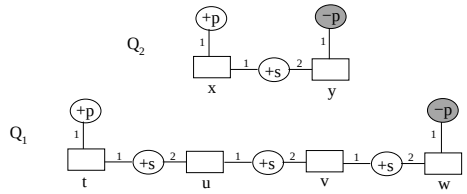
Homomorphism is a core notion in this work. A *homomorphism* h from a PG Q_2 to a PG Q_1 is a mapping from nodes of Q_2 to nodes of Q_1, which preserves bipartition (the image of a term -resp relation- node is a term -resp. relation- node), preserves edges (if rt is an edge with label i in Q_2 then $h(r)h(t)$ is an edge with label i in Q_1), preserves relation node labels (a relation node and its image have the same label) and can instantiate term node labels (if a term node is labeled by a constant, its image has the same label, otherwise there is no constraint on the label of its image). Note that this notion corresponds exactly to the well-known *query homomorphism* defined on positive conjunctive queries; it can be seen as an extension of query homomorphism to negative subgoals.

When there is a homomorphism h from Q_2 to Q_1, we say that Q_2 *maps* to Q_1 by h. Q_2 is called the *source* graph and Q_1 the *target* graph. If Q_2 and Q_1 have only positive subgoals, $Q_1 \sqsubseteq Q_2$ *iff* Q_2 maps to Q_1. When we also consider 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 w.r.t. relation p, we obtain the union of four queries $q_{1,1} = ans() \leftarrow p(t) \wedge s(t,u) \wedge s(u,v) \wedge s(v,w) \wedge \neg p(w) \wedge p(u) \wedge p(v)$, $q_{1,2} = ans() \leftarrow p(t) \wedge s(t,u) \wedge s(u,v) \wedge s(v,w) \wedge \neg p(w) \wedge \neg p(u) \wedge p(v)$, $q_{1,3} = ans() \leftarrow p(t) \wedge s(t,u) \wedge s(u,v) \wedge s(v,w) \wedge \neg p(w) \wedge p(u) \wedge \neg p(v)$ and $q_{1,4} = ans() \leftarrow p(t) \wedge s(t,u) \wedge s(u,v) \wedge s(v,w) \wedge \neg p(w) \wedge \neg p(u) \wedge \neg p(v)$. Each of the queries is a 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 *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.



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

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

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.*

We can further restrict the set of relations considered to those appearing in opposite subgoals both in Q_2 and in Q_1 [LM07]. In the sequel, this set is called the *completion vocabulary* of Q_2 and Q_1 and denoted $\mathcal{V}$.

Let us outline previous algorithmic proposals for checking containment of $CQ^\neg$ queries in this framework. Although it is expressed in a different framework, the first proposal [Ull97] can be recast as follows (see [LM07] for more details): it consists of computing the set of total completions of Q_1 and checking the existence of a homomorphism from Q_2 to each of them (say Q_1^c). The complexity of this algorithm is prohibitive: $\mathcal{O}(2^{(n_{Q_1})^k \times |\mathcal{V}|} \times hom(Q_2, Q_1^c))$, where n_{Q_1} is the number of term nodes in Q_1, k is the maximum arity of a relation, $\mathcal{V}$ is the completion vocabulary 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.

Two kinds of improvements are defined in [WL03] and [LM07]: first, some necessary conditions for containment are exhibited, which can be used to tentatively detect a failure before generating completions; secondly, completions can be incrementally built and checked.

In [WL03], the following necessary but not sufficient condition for containment is exhibited (for *safe* queries but it remains true for general $CQ^\neg$): if $Q_1 \sqsubseteq Q_2$ then there must be a homomorphism, say h, from the positive part of Q_2, say Q_2^p, to Q_1; moreover, this homomorphism should not contradict the negative subgoals of Q_2: for all subgoals $-r(t_1, \ldots, t_k)$ in Q_2, Q_1 should not contain $+r(h(t_1), \ldots, h(t_k))$. This property can be used as a filter: if there is no such homomorphism from Q_2^p to Q_1, then $Q_1 \not\sqsubseteq Q_2$. It is generalized in [LM07] with the notion of *pure subgraphs* and *compatible* homomorphism, as detailed below. Then we have: if $Q_1 \sqsubseteq Q_2$ then, for each *pure* subgraph Q_2' of Q_2, there must be a *compatible* homomorphism from Q_2' to Q_1 w.r.t. Q_2. In the following definitions, we add some notions and notations that we will use in the sequel of this paper.

Definition 2 (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 sub-

[2] Homomorphism checking is NP-complete. A brute-force algorithm solves it in $\mathcal{O}(n_{Q_1}^{n_{Q_2}})$, where n_{Q_2} is the number of term nodes in Q_2.

graph *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.*

We will use the following notations for pure subgraphs of Q_2:

- Q_2^{max} denotes a pure subgraph that is maximal for inclusion;
- Q_2^+ is the Q_2^{max} with all positive relation nodes in Q_2;
- Q_2^- is the Q_2^{max} with all negative relation nodes in Q_2;
- Q_2^{Max} denotes a Q_2^{max} of maximal cardinality.

Note that if a relation label $+r$ or $-r$ in Q_2 does not appear in Q_1 then $Q_1 \not\sqsubseteq Q_2$. Thus, we assume in the following that all relation labels in Q_2 appear in Q_1, which implies that the subgraph induced by the relations not in $\mathcal{V}$ is pure, and all Q_2^{max} contain it. Hence, we have Q_2^p (the positive part of Q_2) $\subseteq Q_2^+$ but the contrary is generally false.

Example 2. See Figure 1: there are two Q_2^{max} (which are also of maximal cardinality): the first one is Q_2^+, which contains $+p(x)$ and $+s(x, y)$, and the second one is Q_2^-, which contains $-p(x)$ and $+s(x, y)$ (because $\mathcal{V} = \{p\}$, thus $s \notin \mathcal{V}$).

Intuitively, a homomorphism from a pure 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 3 (Border, Compatible homomorphism). *Let Q_2 and Q_1 be two PGs and Q_2' be a pure subgraph of Q_2. The relation nodes of $Q_2 \setminus Q_2'$ are called* border relation nodes *of Q_2' w.r.t. Q_2. A homomorphism h from Q_2' to Q_1 is said to be* compatible *w.r.t. Q_2 if, for each border relation node inducing the subgoal $\sim r(t_1, \ldots, t_k)$, the opposite subgoal $\overline{\sim r}(h(t_1), \ldots, h(t_k))$ is* not *in Q_1 and for each pair of opposite border relation nodes respectively on $(c_1, \ldots, c_k)$ and $(d_1, \ldots, d_k)$, $(h(c_1), \ldots, h(c_k)) \neq (h(d_1), \ldots, h(d_k))$.*[4]

Now, let us consider the search space leading from Q_1 to its total completions and partially ordered by the relation "subgraph of". In [LM07], this space is explored as a binary tree with Q_1 as root. The children of a node are obtained by adding, to the graph associated with this node (say Q_1'), a relation node in positive and negative form (each of the two new graphs is thus obtained by a completion step from Q_1'). The aim is to find a set of partial completions covering the set of total completions of Q_1, i.e. the question becomes: "is there a set of partial completions $\{Q_{1,1}, \ldots, Q_{1,n}\}$ of Q_1 such that (1) Q_2 maps to each $Q_{1,i}$ for $i = 1 \ldots n$; (2) each total completion Q_1^c of Q_1 is covered by a $Q_{1,i}$ (i.e. $Q_{1,i}$ is a subgraph of Q_1^c) ?". After each completion step, it is checked whether Q_2 maps to the current partial completion: if yes, this completion is one of the sought $Q_{1,i}$, otherwise the exploration continues. Figure 2 illustrates this method on the very easy Example 1. Two graphs $Q_{1,1}$ and $Q_{1,2}$ are built from Q_1, respectively by adding

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

[4] The last condition is necessary to ensure that a compatible homomorphism from Q_2' to Q_1 can be extended to a homomorphism from Q_2 to a total completion of Q_1. However, it is necessarily satisfied if Q_2' is a pure subgraph that is maximal for inclusion. We only need it for the second approach presented in this paper.

$+p(v)$ 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 $Q_{1,2}$: two graphs $Q_{1,3}$ and $Q_{1,4}$ are built from $Q_{1,2}$, 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). Algorithm 1 implements this method (the numbers in the margin are relative to the refinements studied in Section 3.2).

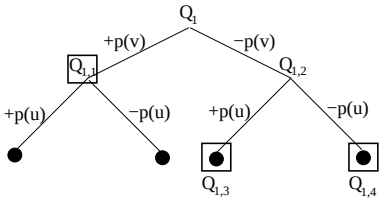


Fig. 2. The search tree of Example 1. Each black dot represents a Q_1^c and each square a $Q_{1,i}$.

Algorithm 1: **recCheck**(Q_1)

Input: a consistent PG Q_1
Data: Q_2, $\mathcal{V}$
Result: true if $Q_1 \sqsubseteq Q_2$, false otherwise
begin
  if *there is a homomorphism from* Q_2 *to* Q_1 **then return** *true* ;
  if Q_1 *is complete w.r.t.* $\mathcal{V}$ **then return** *false* ;
(3) *** *Filtering step* ***\
(1) Choose $r \in \mathcal{V}$ and $t_1, \ldots, t_k$ in Q_1 such that $+r(t_1, \ldots, t_k) \notin Q_1$ and $-r(t_1, \ldots, t_k) \notin Q_1$;
  Let Q'_1 be obtained from Q_1 by adding $+r(t_1, \ldots, t_k)$;
  Let Q''_1 be obtained from Q_1 by adding $-r(t_1, \ldots, t_k)$;
(2) **return** ***recCheck***(Q'_1) *AND* ***recCheck***(Q''_1) ;
end

The algorithm proposed in [WL03] can be seen as exploring the same space of graphs but in a radically different way. At each step, it generates all homomorphisms from Q_2^p to the current Q_1. Then, for each compatible homomorphism in this set, say h, and for each negative subgoal $-p(t_1, \ldots, t_k)$ in Q_2 that cannot be mapped to Q_1 by extending h, a new query to test is generated from Q_1 by adding the positive subgoal $+p(h(t_1), \ldots, h(t_k))$; intuitively, the idea is that each total completion of Q_1 contains either $+p(h(t_1), \ldots, h(t_k))$ or $-p(h(t_1), \ldots, h(t_k))$; if $-p(h(t_1), \ldots, h(t_k))$ were present, then h could be extended to $-p(h(t_1), \ldots, h(t_k))$, thus it remains to test the $+p(h(t_1), \ldots, h(t_k))$ case. In Example 1 (see also Figure 1), only one homomorphism can be found at each step: the homomorphism $\{x \mapsto t, y \mapsto u\}$ from Q_2^p to Q_1 leads to Q'_1 obtained by adding $+p(u)$; at the next step, $\{x \mapsto u, y \mapsto v\}$ from Q_2^p to Q'_1 leads to Q''_1 obtained by adding $+p(v)$; finally, there is a homomorphism from Q_2^p to Q''_1 that can be extended to the negative subgoal, thus no new graph is generated. This algorithm can be seen as developing an *and/or* tree: a homomorphism h leads to success if all queries Q'_i directly generated from it lead to containment; a query Q'_i leads to containment if there is a homomorphism from Q_2^p to $Q_i'^p$ leading to success.

The and/or tree is traversed in a breadth-first manner. Contrarily to Algorithm 1, partial completions built by this algorithm do not partition the space (basically because only positive subgoals are generated), which leads to the problem of detecting that a newly generated graph is not the same as a graph already generated (see the discussion in [LM07] for more details). In this paper, we focus in refining the algorithm in [LM07]. The experimental comparison of both ways of exploring the space of graphs remains to be done.

3 Experimental methodology and algorithm refinements

In this section, we briefly present our experimental methodology, then we propose and analyze three refinements of Algorithm 1.

3.1 Methodology

Due to the lack of benchmarks or real-world data available for the studied problem, we built a random generator of polarized graphs. The chosen parameters are as follows:

- the number of *term* nodes (i.e. the number of terms in the associated query)[5];
- the number of distinct *relations*;
- the *arity* of these relations (set at 2 in the following experiments);
- the *density* per relation, which is, for each relation r, the ratio of the number of subgoals with relation r in the graph to the number of subgoals with relation r in a total completion of this graph w.r.t. $\{r\}$.
- the *percentage of negation* per relation, which is, for each relation r, the percentage of *negative* subgoals with relation r among all subgoals with relation r.

An instance of CQC$^\neg$ is obtained by generating a pair (Q_1, Q_2) of PGs corresponding to *safe* queries. In this paper, we chose the same number of term nodes and the same percentage of negation for both graphs. In the sequel we adopt the following notations: *nbT* represents the number of term nodes, *nbR* the number of distinct relations and *SD* (resp. *TD*) the Source (resp. Target) graph Density per relation. The difficulty of the problem led us to restrict the value of *nbT* to between 5 and 8 (5 for the first experiments, 8 after improvement 1 which greatly decreases the running time).

In order to discriminate between different techniques, we first experimentally studied the influence of the parameters on the "difficulty" of instances. We measured the difficulty in three different ways: the running time, the size of the search tree and the number of homomorphism checks (see [BLM10] for more details). Concerning the *negation percentage*, we checked that the maximal difficulty is obtained when there are as many negative relation nodes as positive relation nodes. One can expect that increasing the number of relations occurring in graphs increases the difficulty, in terms of running time as well as the size of the searched space. Indeed, the number of completions increases exponentially (there are $(2^{{n_{Q_1}}^2})^{nbR}$ total completions for *nbR* binary relations). These intuitions are only partially validated by the experiments: see for instance Table

[5] We do not generate constants; indeed, constants tend to make the problem easier to solve because there are fewer potential homomorphisms; moreover, this parameter does not influence the studied heuristics.

1, which shows, for each number of relations, the density values at the difficulty peak. We observe that the difficulty increases up to a certain number of relations (3 here, with a CPU time of 14809 and a Tree size of 216911) and beyond this value, it continuously decreases. Moreover, the higher the number of relations, the lower the *SD* that entails the greatest difficulty peak, and the higher the difference between *SD* and *TD* at the difficulty peak. In following experiments, we always take the *SD* and *TD* values corresponding to a difficulty peak.

nbR	SD	TD	CPU time (ms)	Tree size
1	0.24	0.24	19	82
2	0.12	0.24	7168	111540
3	0.08	0.4	14809	216911
4	0.08	0.68	12793	119911
5	0.08	0.8	4556	42566

Table 1. Influence of the number of relations (*nbT*=5).

For each value of the varying parameter, we considered 500 instances and computed the mean search cost of the results on these instances (with a timeout set at 5 minutes). The program is written in Java. The experiments were performed on a Sun fire X4100 Server AMD Opteron 252, equipped with a 2.6 GHz Dual-Core CPU and 4G of RAM, under Linux. In the sequel we only show the CPU time when the three difficulty measures are correlated.

3.2 Refinements

We now analyze three refinements of Algorithm 1, which concern the following aspects:

1. the choice of the next subgoal to add;
2. the choice of the child to explore first;
3. dynamic filtering at each node of the search tree.

1. Since the search space is explored in a depth-first manner, the choice of the next subgoal to add, i.e. $\sim r(t_1, \ldots, t_k)$ in Algorithm 1 (Point 1), is crucial. A brutal technique consists of choosing r and $t_1, \ldots, t_k$ randomly. Our proposal is to guide this choice by a compatible homomorphism, say h, from a Q_2^{max} to the current Q_1. More precisely, the border relation nodes $\sim r(e_1, \ldots, e_k)$ w.r.t. this Q_2^{max} can be divided into two categories. In the first category, we have the border nodes s.t. $\sim r(h(e_1), \ldots, h(e_k)) \in Q_1$, which can be used to *extend* h; if all border nodes are in this category, h can be extended to a homomorphism from Q_2 to Q_1. The choice of the subgoal to add is based on a node $\sim r(e_1, \ldots, e_k)$ in the second category: r is its relation symbol and $t_1, \ldots, t_k = h(e_1), \ldots, h(e_k)$ are its neighbors (note that neither $\sim r(h(e_1), \ldots, h(e_k))$ nor $\overline{\sim r}(h(e_1), \ldots, h(e_k))$ is in Q_1 since $\sim r(e_1, \ldots, e_k)$ is in the second category and h is compatible). Intuitively, the idea is to give priority to relation nodes potentially able to transform this compatible homomorphism into a homomorphism from Q_2 to a (partial) completion of Q_1, say Q_1'. If so, all completions including Q_1' are avoided.

Figure 3 shows the results obtained with the following choices:

- *random choice*;
- *random choice + filter*: random choice and Q_2^+ as filter (i.e. at each `recCheck` step a compatible homomorphism from Q_2^+ to Q_1 is looked for: if none exists, the false value is returned);
- *guided choice*: Q_2^+ used both as a filter and as a guide.

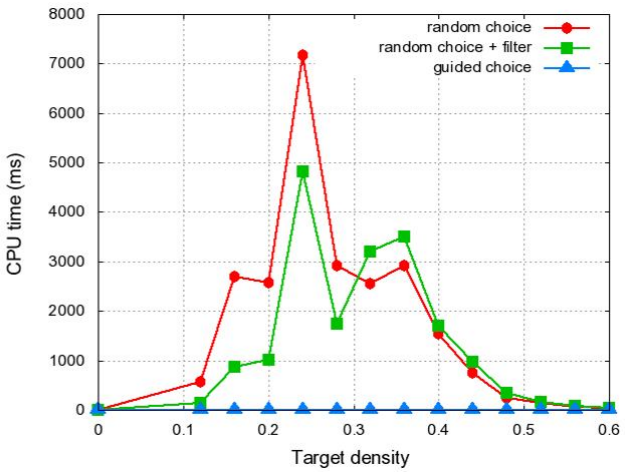


Fig. 3. Influence of the completion choice : *nbT*=5, *nbR*=2, *SD*=0.12

Note that the guided choice comes with an implicit filter: indeed, when a compatible homomorphism from Q_2^+ to a partial completion of Q_1 (say Q_1') is sought, the false value is returned if none exists (since $Q_1' \not\sqsubseteq Q_2$). In order to only discriminate choice heuristics, we also considered a random choice with a filter. As expected, the guided choice is always much better than the random choice (with or without filter).

2. Experiments have shown that the order in which the children of a node, i.e. Q_1' and Q_1'' in Algorithm 1 (Point 2), are explored is important. Assume that Point 1 in Algorithm 1 relies on a guiding subgraph. Consider Figure 4, where Q_2^+ is the guiding subgraph (hence the border is composed of negative relation nodes), "Extension" means "Q_1'' first" and "Contradiction" means the reverse order: we see that it is always better to explore Q_1' before Q_1''. If we take Q_2^- as the guiding subgraph, then the inverse order is better. More generally, let $\sim r(e_1, \ldots, e_k)$ be the border node that defines the subgoal to add. Let us call *h-extension* (resp. *h-contradiction*) the graph built from Q_1 by adding $\sim r(h(e_1), \ldots, h(e_k))$ (resp. $\overline{\sim r}(h(e_1), \ldots, h(e_k))$). See Example 3. It is better to first explore the child corresponding to the *h-contradiction*. Intuitively, by contradicting the compatible homomorphism found, this gives priority to failure detection.

Example 3. See Figure 1. $Q_2^+ = \{+p(x), +s(x, y)\}$. Let Q_2^+ be the guiding subgraph. The only border node of Q_2^+ w.r.t. Q_2 is $-p(y)$. $h = \{x \mapsto t, y \mapsto u\}$ is the only compatible homomorphism from Q_2^+ to Q_1. The h-extension (resp. h-contradiction) is obtained by adding $+p(u)$ (resp. $-p(u)$).

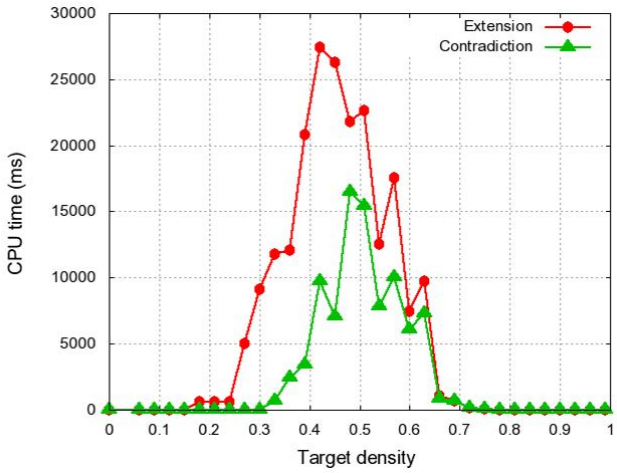


Fig. 4. Influence of the exploration order : *nbT=8*, *nbR=3*, *SD=0.06*

3. The last improvement consists of performing dynamic filtering at each node of the search tree. Once again, the aim is to detect a failure sooner. More precisely, we consider a set of Q_2^{max} and check if there is a compatible homomorphism from each element in this set to the newly generated graph. Table 2 shows the obtained results (at a difficulty peak) with the following configurations:

- *Max*: Q_2^{Max} as guide and no filter (other than Q_2^{Max});
- *Max-*$\overline{Max}$: Q_2^{Max} as guide and $Q_2^{\overline{Max}}$ (the subgraph on the relation nodes in $Q_2 \setminus Q_2^{Max}$) as filter;
- *Max-all*: Q_2^{Max} as guide and all Q_2^{max} as filters.

Unsurprisingly, the stronger the dynamic filtering, the smaller the size of the search tree. The CPU time is a bit higher for *Max-all* but this configuration checks much more homomorphisms than the others. Since our current algorithm for (compatible) homomorphism checking can be considerably improved, these results show that *Max-all* is the best choice.

Configuration	CPU time (ms)	Tree size	Hom check
Max	7404	3810	5539
Max-$\overline{Max}$	7421	3765	6209
Max-all	8427	2249	13331

Table 2. Influence of the dynamic filtering: *nbT*=8, *nbR*=3, *SD*=0.06, *TD*=0.54

The algorithm finally obtained is called `recCheckPlus` and it is shown in Algorithm 2. It is initially called with $(Q_1, \emptyset)$. The second parameter is used to memorize the compatible homomorphism found for the father of the current node, in the case where this node is an h-extension of its father (see Q_1'' in the algorithm); otherwise, the compatible homomorphism for its father has been contradicted and a new one has to be computed, which is done in the *chooseCompletionSubgoal* subalgorithm. More

precisely, this subalgorithm returns a completion literal as explained in Point 1 and a new compatible homomorphism h if needed.

Algorithm 2: **recCheckPlus**(Q_1, h)

Input: a consistent PG Q_1 and a compatible homomorphism h from the guiding subgraph to the father of Q_1 (empty for the root)
Data: Q_2, $\mathcal{V}$
Result: true if $Q_1 \sqsubseteq Q_2$, false otherwise
begin
  if *there is a homomorphism from* Q_2 *to* Q_1 **then return** *true* ;
  if Q_1 *is complete w.r.t.* $\mathcal{V}$ **then return** *false* ;
(3) **if** ***dynamicFiltering***(Q_1) = *failure* **then return** *false* ;
(1) $l, h \leftarrow$ **chooseCompletionSubgoal**(Q_1, h) ;
  Let Q_1' be obtained from Q_1 by adding $\overline{l}$;
  Let Q_1'' be obtained from Q_1 by adding l ;
(2) **return** ***recCheckPlus***($Q_1', \emptyset$) *AND* ***recCheckPlus***(Q_1'', h) ;
end

4 Second approach

In this section, we present the second method, which consists of translating the CQC$^\neg$ problem into the problem of checking unsatisfiability of a propositional formula in conjunctive normal form (*i.e.* a conjunction of disjunctions of propositional literals), called the UNSAT problem. This method is then experimentally compared to `recCheckPlus`.

4.1 Method

Let us first explain the main ideas of this method. Instead of exploring the space of graphs in a depth-first manner in order to find a set of partial completions that covers all total completions, a *candidate* covering set is built at once; it is built from all compatible homomorphisms from a specific pure subgraph of Q_2 to Q_1; this candidate is indeed a covering set if and only if a formula built from it is valid; this formula is built by considering for each compatible homomorphism the relation nodes that should be added to Q_1 to obtain a homomorphism from Q_2. More specifically, we proceed in three steps:

1. Compute all compatible homomorphisms from Q_2' (the special subgraph of Q_2) to Q_1;
2. Build a propositional formula F_{prop}, that is the disjunction of, for each compatible homomorphism h of Step 1, the conjunction C of missing subgoals in Q_1 for h to be a homomorphism from Q_2 to $Q_1 \wedge C$;
3. Check if $\overline{F_{prop}}$ (an UNSAT instance) is unsatisfiable: if yes, then $Q_1 \sqsubseteq Q_2$, otherwise $Q_1 \not\sqsubseteq Q_2$.

Step 1: Compute the candidate covering set. The method is based on a specific pure subgraph of Q_2, which necessarily maps to the Q_1 part of each total completion of Q_1 when $Q_1 \sqsubseteq Q_2$ (note that this is not true for all pure subgraphs of Q_2).

Definition 4 (Stable subgraph). *The* stable subgraph *of Q_2, denoted Q_2^s, contains all term nodes in Q_2 and all subgoals in Q_2 with a relation that does not appear in the completion vocabulary of Q_1 and Q_2 (note that Q_2^s is included in all Q_2^{max}).*

Since the stable subgraph does not contain relations belonging to subgoals added to Q_1 during completion, we have the desired property:

Property 1. Let Q_1 and Q_2 be two PGs. If h is a homomorphism from Q_2 to a total completion of Q_1, then h is a compatible homomorphism from Q_2^s to Q_1.

Furthermore, we can replace, without incidence on the CQC$^\neg$ problem, each variable in Q_1 by a **new** constant. This modification preserves all homomorphisms to Q_1. In the sequel, we consider that Q_1 contains only constants. In particular, the query q_1 in Example 1 becomes: $q_1 = ans() \leftarrow p(a) \wedge s(a,b) \wedge s(b,c) \wedge s(c,d) \wedge \neg p(d))$. This will allow to obtain a propositional formula in Step 2.

Example 4. Let Q_1 and Q_2 be PGs of Figure 1. $Q_2^s = s(x,y)$. There are 3 compatible homomorphisms from Q_2^s to Q_1 : $h_1 = \{x \mapsto a, y \mapsto b\}$, $h_2 = \{x \mapsto b, y \mapsto c\}$ and $h_3 = \{x \mapsto c, y \mapsto d\}$.

Step 2: Build the propositional formula. With each compatible homomorphism computed at Step 1, we build a conjunction of the "missing" subgoals in Q_1. Each partial completion of Q_1 obtained by adding these subgoals to Q_1 is an element of the candidate covering set.

Definition 5 (Minimal conjunction). *The* minimal conjunction *of Q_1 w.r.t. a compatible homomorphism h, denoted C^m, is the conjunction composed of the atom* ■[6] *and the subgoals $\sim r(h(t_1), \ldots, h(t_k))$ such that $\sim r(t_1, \ldots, t_k) \in (Q_2 \setminus Q_2^s)$ and $\sim r(h(t_1), \ldots, h(t_k)) \notin Q_1$.*

Example 5. For $h_1 : C_1^m = \neg p(b)$; for $h_2 : C_2^m = p(b) \wedge \neg p(c)$; for $h_3 : C_3^m = p(c)$.

Then we build the entire formula, which is the disjunction of all minimal conjunctions:

Definition 6 (Disjunction of Stable Minimal Conjunctions ($\mathcal{DSMC}$)). *We call $\mathcal{DSMC}(Q_2, Q_1)$ the disjunction of the atom* □[7] *and all minimal conjunctions w.r.t. the compatible homomorphisms from Q_2^s to Q_1 (i.e. $\square \vee C_1^m \vee \ldots \vee C_i^m$, denoted $\bigvee C^m$).*

Example 6. $\mathcal{DSMC}(Q_2, Q_1) = \neg p(b) \vee (p(b) \wedge \neg p(c)) \vee p(c)$.

The theorem validating this approach is the following (Q_1 is assumed to be consistent):

Theorem 2. *$Q_1 \sqsubseteq Q_2$ iff $\mathcal{DSMC}(Q_2, Q_1)$ is valid.*

The following definitions and lemmas are used to prove the theorem.

[6] ■ is the tautology: it is necessary when h is a homomorphism from Q_2 to Q_1, otherwise the conjunction would be empty.

[7] □ is the absurd literal: it is necessary when there is no compatible homomorphism from Q_2^s to Q_1, otherwise the disjunction would be empty.

Definition 7 (Total conjunction). *Let Q_1^c be a total completion of Q_1. The* total conjunction *of Q_1^c, denoted C, is the conjunction of all subgoals $\sim r(t_1, \ldots, t_k) \in (Q_1^c \setminus Q_1)$.*

Definition 8 (Disjunction of Total Conjunctions ($\mathcal{DTC}$)). *We call $\mathcal{DTC}(Q_1)$ the disjunction of all total conjunctions of Q_1 (i.e. $C_1 \vee \ldots \vee C_j$).*

Lemma 1. *$\mathcal{DTC}(Q_1)$ is valid.*

Lemma 2. *If $Q_1 \sqsubseteq Q_2$ then for all $C \in \mathcal{DTC}(Q_1)$, there is a $C^m \in \mathcal{DSMC}(Q_2, Q_1)$ such that $C^m \subseteq C$ (i.e. all subgoals in C^m are also in C).*

Property 2. Let C^m be a minimal conjunction of Q_2 w.r.t. a compatible homomorphism from Q_2^s to Q_1. $Q_1 \wedge C^m \sqsubseteq Q_2$.

Proof of Theorem 2:

$\Longleftarrow$ Since $\mathcal{DSMC}(Q_2, Q_1)$ is valid, $Q_1 \equiv Q_1 \wedge \bigvee C^m \equiv \bigvee (Q_1 \wedge C^m)$. According to Property 2, $\bigvee (Q_1 \wedge C^m) \sqsubseteq Q_2$. Thus $Q_1 \sqsubseteq Q_2$.

$\Longrightarrow$ Let $Q_1 \sqsubseteq Q_2$. According to Lemma 2, for all $C \in \mathcal{DTC}(Q_1)$, there is $C^m \in \mathcal{DSMC}(Q_2, Q_1)$ s.t. $C = C^m \wedge C'$ where C' is a conjunction of subgoals. *By absurd*: Assume $\mathcal{DSMC}(Q_2, Q_1)$ is not valid. Then there is an interpretation $\mathcal{I}$ s.t. for all $C^m \in \mathcal{DSMC}(Q_2, Q_1)$, $\mathcal{I} \models \neg C^m$.Thus for all $C \in \mathcal{DTC}(Q_1)$, $\mathcal{I} \models \neg C$.Thus $\mathcal{DTC}(Q_1)$ is not valid, which contradicts Lemma 1. Hence $\mathcal{DSMC}(Q_2, Q_1)$ is valid. □

Step 3: Translate into UNSAT. The negation of $\mathcal{DSMC}(Q_2, Q_1)$ is a propositional conjunctive normal form, which enables us to use a SAT solver.

Example 7. $CF = \overline{\mathcal{DSMC}(Q_2, Q_1)} = p(b) \wedge (\neg p(b) \vee p(c)) \wedge \neg p(c)$. CF is unsatisfiable, thus $\mathcal{DSMC}(Q_2, Q_1)$ is valid and $Q_1 \sqsubseteq Q_2$.

4.2 Algorithm and experiments

Algorithm 3: UNSATCheck(Q_1)

```
Input: a consistent PG Q1
Data: Q2, Q2^s
Result: true if Q1 ⊑ Q2, false otherwise
begin
    DSMC(Q2, Q1) ← □ ;
    h1, ..., hn ← findAllCompatibleHomomorphisms(Q2, Q2^s, Q1) ;
    foreach hi, i = 1...n do
        foreach border node ~r(t1, ..., tk) ∈ Q2 \ Q2^s do
            Ci^m ← ■ ;
            if ~r(hi(t1), ..., hi(tk)) ∉ Q1 then
                Ci^m ← Ci^m ∧ ~r(hi(t1), ..., hi(tk)) ;
        DSMC(Q2, Q1) ← DSMC(Q2, Q1) ∨ Ci^m ;
    return UNSAT(¬DSMC(Q2, Q1))
end
```

Algorithm 3 implements this method. The UNSAT call uses the well-known *Sat4J* solver[8].

[8] http://www.sat4j.org/

To compare UNSATCheck and recCheckPlus, we used the size of the vocabulary completion (directly correlated with the size of the stable subgraph) as the varying parameter. Indeed, this parameter is crucial for UNSATCheck (note that this parameter has also an influence for recCheckPlus): the bigger the stable subgraph, the lower the number of compatible homomorphisms and then the size of the obtained formula. We built 1000 random instances for each value of $|\mathcal{V}|$ (for each relation, the percentage of negation is equal to 0%, 50% or 100%) and compared UNSATCheck and recCheckPlus on them: see Table 3.[9]

We observe that for both algorithms the maximal difficulty is for $|\mathcal{V}| = 3$. Then UNSATCheck is the worst because the stable subgraph contains only term nodes, thus the number of compatible homomorphisms and then the size of the obtained formula are exponential in the sizes of the initial queries. As expected, the increase of the size of the stable graph makes the algorithms better. For $|\mathcal{V}| = 2$, UNSATCheck is a little better than recCheckPlus. When the size of the stable graph is the highest, the results for both algorithms are similar. These results show that the choice of an algorithm rather than another depends on the size of the stable graph. These are only preliminary results, further experiments are needed to refine this choice.

Size of $\mathcal{V}$	Size of the stable subgraph	recCheckPlus CPU time	UNSATCheck CPU time			
			Total	1st step	2nd step	3rd step
3	0	6482	27038	10541	15378	1119
2	6	800	444	172	229	43
1	12	5	20	9	2	9
0	18	1	2	2	0	0

Table 3. Detailed comparison of the two algorithms : *nbT*=8, *nbR*=3, *SD*=0.06, *TD*=0.51.

5 Perspectives

In this paper, we have refined the algorithm proposed in [LM07] and checked experimentally several choices. These refinements heavily rely on special subgraphs, called pure subgraphs. Using pure subgraphs of maximal cardinality as guiding and filtering graphs seems a good choice. However, the size of pure subgraphs is not the "ultimate" criterion, as shown in Figure 5: for each instance, we ran recCheckPlus with all possible Q_2^{Max} (they all have the same size). The Maximum (resp. Minimum) curve is obtained by choosing, for each instance, the worst (resp. best) Q_2^{Max}, i.e. that leads to the highest (resp. lowest) CPU time. We conclude that the choice of the Q_2^{Max} used to guide among all Q_2^{Max} is a determining step. However, finding criteria allowing to better discriminate between pure graphs is an open issue.

In [WL03] another way of exploring the query space is proposed. The associated algorithm is much more complex to follow and implement than recCheck. Furthermore, some parts were not specified (for instance how to avoid generating a query that was already generated). We are currently implementing this algorithm, while integrating the improvements designed for recCheck.

[9] Note that 8 term nodes is the highest value that UNSATCheck can deal with (with $|\mathcal{V}| = 3$ and our implementation): beyond 8, the memory space explodes.

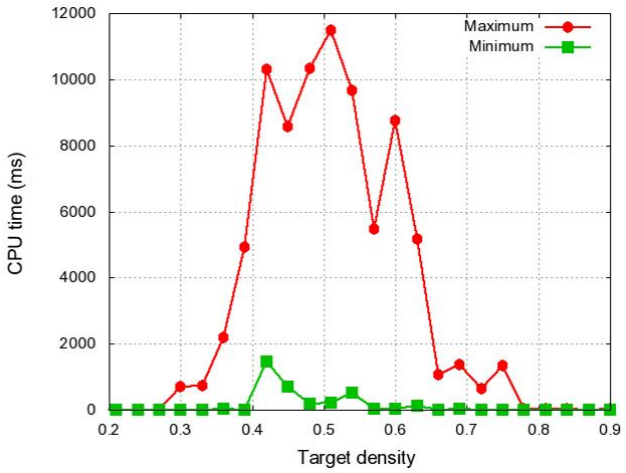


Fig. 5. Comparison between the worst and the best choice of Q_2^{Max}: *nbT*=8, *nbR*=3, *SD*=0.06

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.

[BLM10] 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.

[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.

[Sai08] L. Sais. *Problème SAT : progrès et défis*. Hermes, 2008.

[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.