

A Unified Apriori-like Algorithm for Conjunctive Query Containment

Fang Wei
University of Freiburg
fwei@informatik.uni-freiburg.de

Georg Lausen
University of Freiburg
lausen@informatik.uni-freiburg.de

ABSTRACT

The design of containment checking algorithms for conjunctive queries with arithmetic comparisons and safe negations (CQ^{\neq} s) is fundamental to many database applications. However, it is a challenging task due to the intractability of the problem. Existing algorithms are either computationally too expensive, or capable to deal with only a fragment of these extensions.

In this paper we propose a novel algorithm for testing containment of CQ^{\neq} s. The key idea of the algorithm is to recursively consider only the containment mappings between the positive subgoals of the queries, instead of testing all the symbol mappings. Thus the number of test cases can be drastically reduced. Our algorithm enables further an on the fly execution of the *normalization* step.

With the observation that the property of *anti-monotonicity* holds in our problem setting, we develop an *Apriori*-like algorithm as a sub-function of the containment checking algorithm, to which desirable pruning strategies are applied.

Furthermore, we identify the criteria to reduce the number of test sets, and demonstrate that two additional improvements to the containment checking algorithm: *node clustering* and *tuple pruning* can further speed up the checking process.

1. INTRODUCTION

Conjunctive query (CQ) containment checking is a classical problem in database research. Besides the traditional role it plays in semantic optimization, query containment has been shown crucial in data integration [9], data exchange [6], and semantic web [14], just to mention a few.

Since it has been proved that the containment checking problem of CQ s is NP-complete [5], many research works have been focused on extracting fragments of CQ for which the containment checking is tractable. Specifically, it has been shown that if the containing query is *acyclic*, then a linear time algorithm can be obtained for the containment checking [21]. Moreover, if each database predicate does not occur more than twice in the body of the contained query, the containment checking is tractable as well [16]. Recently,

Gottlob et al. [7] have proposed the concept of hypertree width, which is a generalization of acyclicity. If the containing query has bounded hypertree width, the containment checking can be performed in polynomial time.

All of the above results, however, consider only the SPJ (selection, projection and join) queries. Often in real applications, users wish to query the database with arithmetic comparisons like *salary > 10k*, as well as with some safe negations (*not exists* in SQL). We denote such extensions of the CQ as conjunctive query with arithmetic comparisons (CQ^{\neq}), conjunctive query with safe negations (CQ^{\neg}), and conjunctive query containing both arithmetic comparisons and safe negations ($CQ^{\neq,\neg}$).

Containment checking for CQ s with such extensions is harder. It has been proved that the complexity of containment checking for CQ^{\neq} s is Π_2^2 -complete [10, 19]. Furthermore, Kolaitis et al. have shown that even if the positive subgoals of the containing query constitute an acyclic hypergraph, the complexity remains Π_2^2 -hard [11]. Therefore, it is a challenging task to design practical containment checking algorithms for the CQ s with these extensions.

For CQ^{\neq} s, there does exist an elegant algorithm proposed by Zhang and Özsoyoglu [22], as well as by Gupta et al. [8]. We denote this algorithm as ZOG algorithm. The ZOG algorithm first searches for *all* the containment mappings between the positive subgoals and then proceeds with an implication test of the arithmetic comparisons according to these containment mappings. However, it has been shown that in order to test the containment, both queries have to be first *normalized*. The normalization increases (1) the number of arithmetic comparisons, and (2) the number of containment mappings between the queries. Since the efficiency of the implication test depends on both factors, normalization is not desired. Afrati et al. [2] proposed several fragments of CQ^{\neq} s, where the normalization is not necessary. However, in general the normalization step cannot be avoided.

For CQ^{\neg} s, Wei and Lausen [20] proposed an algorithm which recursively tests the containment mappings of the positive subgoals. Based on this work, some algorithmic improvements were proposed using the concept of graph homomorphism [12]. However, neither of these methods can deal with arithmetic comparisons.

To the best of our knowledge, the only existing algorithm checking containment of CQ^{\neq} s is from Levy and Sagiv [13] (see also the description given by Ullmann [18]). We denote this algorithm as LSU algorithm. The LSU algorithm follows the two consecutive steps: (1) exhaustively generate all the canonical databases of the contained query, and (2) apply the containing query on these canonical databases. Since the number of canonical databases is finite, the containment

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IDEAS08 2008, September 10-12, Coimbra [Portugal]

Editor: Bipin C. DESAI

Copyright 2008 ACM 978-1-60558-188-0/08/09 ...\$5.00.

test always terminates. Unfortunately, this algorithm is not viable in practice, even for queries with relatively small size. For illustration consider the conjunctive queries q_1 and q_2 in Example 1.1:

EXAMPLE 1.1.

$$\begin{aligned} q_1 : \text{ans}() &\leftarrow p(X, Y, Z), p(Z, U, V), \neg p(U, Y, V). \\ q_2 : \text{ans}() &\leftarrow p(A, B, C), p(D, E, F), B \neq F, D \neq E. \end{aligned}$$

According to LSU algorithm, if we only consider one partition: $\{p(0, 1, 2), p(2, 3, 4)\}$, then there exist 5^3 tuples with the form $p(\cdot, \cdot, \cdot)$ over the domain $\{0, 1, 2, 3, 4\}$. Hence the number of all possible canonical databases sums up to $2^{5^3} = 2^{125}$, which is computationally too expensive. Furthermore, it is unclear, how the partitioning of the canonical databases should be done, if the CQ^{\neq} query contains arithmetic comparisons with constants (such as $X < 5$).

1.1 Our Contributions

Despite numerous theoretical results, several problems concerning the containment checking of CQ^{\neq} s are still open:

1. Is it possible to improve the LSU algorithm to obtain an acceptable running time?
2. Could the ZOG algorithm be further improved, so that the normalization step is only executed when necessary?
3. Does a feasible algorithm exist, which can deal with any kind of arithmetic comparisons and safe negation in a uniform way?

In this paper, we propose a novel algorithm for testing containment of CQ^{\neq} s, which positively answers all these questions.

The key observation is as follows: Consider two CQ^{\neq} s q_1 and q_2 . $q_1 \sqsubseteq q_2$ is to be checked. The ultimate goal of our algorithm is finding a more *specific* query q of q_1 , such that there exists a canonical database D of q , and $q_2(D) = \text{FALSE}$. If such a query q exists, we conclude with $q_1 \not\sqsubseteq q_2$, otherwise $q_1 \sqsubseteq q_2$ holds. Therefore, we are interested in the following question: *How to generate all these more specific queries of q_1 efficiently?* We prove that it suffices to systematically extend q_1 by adding the subgoals (or arithmetic comparisons) with the form $\rho_i(\text{co}(cn_j))$ to q_1 , where ρ_i is the *containment mapping* from positive subgoals of q_2 to those of q_1 , cn_j is either a negated subgoal or an arithmetic comparison in the body of q_2 , and $\text{co}(cn_j)$ is the *complement* (see Definition 2.4) of cn_j . This way, (1) we can deal with both arithmetic comparisons and safe negations in a uniform way, and (2) the number of tests can be drastically reduced, comparing with the method of generating all the symbol mappings between two queries.

EXAMPLE 1.2.

$$\begin{aligned} q_1 : \text{ans}() &\leftarrow p(X, Y), \neg p(Y, X). \\ q_2 : \text{ans}() &\leftarrow p(A, B), A \neq B. \end{aligned}$$

Consider Example 1.2, we first decide the containment mappings of the positive subgoals of q_2 to those of q_1 , which is $\rho : A \rightarrow X, B \rightarrow Y$. Then we generate a more specific query of q_1 by adding $\rho(\text{co}(A \neq B))$ (which turns out to be $X = Y$) into q_1 . Now the more specific query q has the form

$$q : \text{ans}() \leftarrow p(X, Y), \neg p(Y, X), X = Y.$$

It is obvious that q is unsatisfiable, because it is equivalent to the query

$$q' : \text{ans}() \leftarrow p(X, Y), \neg p(Y, X), X = Y, p(X, X), \neg p(X, X).$$

which contains contradictory subgoals $p(X, X)$ and $\neg p(X, X)$. This observation inspires our definition of the *complete form* of the queries (see Definition 2.14), where additional subgoals resulting from the introduced variable equivalence (such like $X = Y$) have to be added. The concept of *complete form* of queries not only serves for the syntactic testing of unsatisfiability (as in Example 1.2), but also, which is more important, ensures the existence of a canonical database of the query, which enables an *inverse containment mapping* from the database to the positive subgoals in the query. This property is crucial for proving the main theorem (Theorem 3.1).

The rewriting of a query into its complete form, however, could result in new containment mappings between the positive counterpart of the queries. In such cases, we have to generate the new containment mappings and execute the process recursively. It should be pointed out that the generation of the *complete form* is, to some extent, the reminiscence of the *normalization* step of the ZOG algorithm. However, our method differs from the normalization significantly in that the generation of the complete form is done *on the fly*, i.e., it is created only if necessary. On the contrary, the normalization step of the ZOG algorithm is executed before any containment test starts.

It turns out that the algorithm directly interpreted from the theorem is not optimal. Given a fixed set of containment mappings, an exponential number of more specific queries of q_1 have to be generated and tested. We observe that the so-called *anti-monotonicity* property holds in our problem setting. That is, if a query q is unsatisfiable, then any other specific query extended from q (i.e., by adding subgoals or arithmetic comparisons into the body of it) is unsatisfiable as well. Thus any further extension of this query can be avoided. It is well known that with the anti-monotonicity property, the *Apriori*-like algorithm can be applied. Based on this observation, we construct a level-wise Apriori algorithm, where effective pruning strategies can be applied. Moreover, we prove the *early termination* condition, i.e., during the level wise generation of test cases, if some unsatisfiable condition is fulfilled, the algorithm stops and returns the result.

Moreover, after identifying the criteria to reduce the number of test sets, we propose two additional improvements, namely *node clustering* and *tuple pruning*, for the Apriori algorithm. We demonstrate that by integrating these strategies into our containment checking algorithm, significant reduction on both time and space consumption can be achieved.

Of course, since it is proved that the containment of only CQ^{\neq} s is already Π_2^p -complete, no tractable algorithm for our problem can be obtained unless $P=NP$. In this paper we do not continue research to find tractable fragments of the problem; instead we concentrate on the general case and develop algorithms for which we can expect reasonable performance which make them useful in practical cases. This situation is akin to mining frequent item sets, a problem computationally intractable in general, however in practice still manageable because of the clever design of so-called Apriori algorithms [3].

The rest of the paper is organized as follows. Section 2 introduces the definitions of CQ^{\neq} , query evaluation, query

containment and satisfiability. We state the main theorem (Theorem 3.1) and give the proof in Section 3.1. In Section 3.2 the basic algorithm CONTAINMENT is introduced, which is followed by the correctness and completeness proof of the algorithm w.r.t. Theorem 3.1. In Section 3.3 we propose the APRIORI algorithm as a sub-function for CONTAINMENT, where important pruning strategies are applied. Further in Section 4 we propose two improvements for the APRIORI algorithm, node clustering and tuple pruning. The conclusions are given in Section 5.

2. PRELIMINARIES

A (boolean)¹ conjunctive query (CQ) q is of the form

$$\text{ans}() \leftarrow p_1, \dots, p_n.$$

where each p_i ($1 \leq i \leq n$) is a subgoal of the form $g_i(\bar{X}_i)$ and every predicate argument in \bar{X}_i is either a variable or a constant. We denote all the variables in q as $\text{vars}(q)$ and all the constants as $\text{cons}(q)$.

DEFINITION 2.1 (CQ^{\neq} , CQ^\neq AND CQ^\neg). *A conjunctive query with arithmetic comparison and safe negation (CQ^{\neq}) is of the form*

$$\text{ans}() \leftarrow p_1, \dots, p_n, cn_1, \dots, cn_m.$$

where for some r with $1 \leq r \leq m$,

- cn_i ($1 \leq i \leq r$) has the form $X\theta Y$ or $X\theta a$, where $\theta \in \{<, \leq, >, \geq, \neq, =\}$, X, Y are variables and a is a constant.
- cn_i ($r+1 \leq i \leq m$) has the form $\neg h_i(\bar{Y}_i)$, where each argument in \bar{Y}_i is either a variable or a constant.

We denote p_1, \dots, p_n as q^+ , cn_1, \dots, cn_m as q^{cn} , cn_1, \dots, cn_r as q^c , and cn_{r+1}, \dots, cn_m as q^n .

We assume all the CQ^{\neq} queries are safe, i.e., each variable in cn_1, \dots, cn_m occurs in $\text{vars}(q^+)$.

If a conjunctive query contains only arithmetic comparisons, it is a CQ^\neq ; and if it contains only negated subgoals, it is a CQ^\neg .

2.1 Known Framework

Given a database D , we denote the active domain of D as $\text{adom}(D)$.

DEFINITION 2.2 (SYMBOL MAPPING). *Let q , q_1 and q_2 be CQ^{\neq} s, and D a database.*

- A symbol mapping from q to D is a mapping that maps every constant in $\text{cons}(q)$ to the same constant and every variable in $\text{vars}(q)$ to an element in $\text{adom}(D)$.
- A symbol mapping from q_2 to q_1 is a mapping that maps every constant in $\text{cons}(q_2)$ to the same constant in $\text{cons}(q_1)$ and every variable in $\text{vars}(q_2)$ to either a variable in $\text{vars}(q_1)$ or a constant in $\text{cons}(q_1)$.

DEFINITION 2.3 (CONTAINMENT MAPPING). *Let q , q_1 and q_2 be CQ^{\neq} s, and D a database.*

- A containment mapping ρ from q^+ to D is a symbol mapping from q^+ to D , such that for each $p_i \in q^+$, $\rho(p_i) \in D$ holds. We denote all the containment mappings from q^+ to D as $\text{CMs}(q^+, D)$.

- A containment mapping ρ from q_2^+ to q_1^+ is a symbol mapping from q_2^+ to q_1^+ , such that for each $p_i \in q_2^+$, $\rho(p_i) \in q_1^+$. We denote all the containment mappings from q_2^+ to q_1^+ as $\text{CMs}(q_2^+, q_1^+)$.

2.1.1 Query Evaluation

Given a database D and a conjunctive query q , we use $q(D) = \text{TRUE}$ to denote that q is evaluated TRUE over D .

It is well known that if q is a CQ, then $q(D) = \text{TRUE}$ if there exists a containment mapping from q to D [1]. On the contrary, $q(D) = \text{FALSE}$ if such a containment mapping does not exist.

Before we are able to define the query evaluation for CQ^{\neq} s, we have to introduce first the complement operation. Roughly speaking, the complement of a negated subgoal is its positive counterpart, and the complement of an arithmetic comparison expression is the corresponding expression with the complement operation.

DEFINITION 2.4 (COMPLEMENT). *Let q be a CQ^{\neq} and cn_i be either an arithmetic comparison in q , or a negated subgoal. The complement of cn_i , denoted as $\text{co}(cn_i)$ is defined as follows:*

- If cn_i is an arithmetic comparison with the form $X\theta Y$ or $X\theta a$, the complement of cn_i is an arithmetic comparison with the form $X\bar{\theta}Y$ or $X\bar{\theta}a$ where $\bar{\theta}$ is the complement operation of θ .
- If cn_i is a negated subgoal of the form $\neg h_i(\bar{Y}_i)$, the complement of cn_i is the non-negated subgoal $h_i(\bar{Y}_i)$.

For instance, $\text{co}(X \neq a)$ is $X = a$ and $\text{co}(\neg g(X, Y))$ is $g(X, Y)$.

DEFINITION 2.5 (QUERY EVALUATION OF CQ^{\neq}). *Let D be a database and q a CQ^{\neq} of the form*

$$\text{ans}() \leftarrow p_1, \dots, p_n, cn_1, \dots, cn_m.$$

where for some $r \leq m$, every cn_i ($1 \leq i \leq r$) is an arithmetic comparison and every cn_i ($r+1 \leq i \leq m$) is a negated subgoal.

$q(D) = \text{TRUE}$ if and only if there exists a containment mapping ρ from q^+ to D , such that for every cn_i ($1 \leq i \leq r$), $\rho(cn_i)$ is TRUE, and for every cn_i ($r+1 \leq i \leq m$), $\rho(\text{co}(cn_i)) \notin D$.

$q(D) = \text{FALSE}$ if and only if for every containment mapping $\rho \in \text{CMs}(q^+, D)$, there exists a cn_i ($1 \leq i \leq r$), such that $\rho(\text{co}(cn_i))$ is TRUE, or there exists a cn_i ($r+1 \leq i \leq m$), such that $\rho(\text{co}(cn_i)) \in D$.

2.1.2 Containment Checking

Let q_1 and q_2 be conjunctive queries. q_1 is contained in q_2 , denoted as $q_1 \sqsubseteq q_2$, if and only if for every database D , such that $q_1(D)$ is TRUE, $q_2(D)$ is TRUE. Let q_1 and q_2 be CQs. $q_1 \sqsubseteq q_2$ if and only if there is a containment mapping from q_2 to q_1 .

Considering CQ^{\neq} s, a single containment mapping does not suffice. Instead, we need to check a set of containment mappings.

DEFINITION 2.6 (NORMALIZATION). [22] *A CQ^\neq q is normalized if every variable occurs only once in q^+ and there does not exist any constant in q^+ .*

Any CQ^\neq query can be transformed into an equivalent normalized query as follows: (1) For all occurrences of a

¹We consider in this paper only boolean queries.

Table 1: Symbols used in the paper

Symbol	Meaning
CQ	Conjunctive query
CQ^\neq	CQ with arithmetic comparison
CQ^\neg	CQ with safe negation
CQ^\neq^\neg	CQ with arithmetic comparison and safe negation
q^+	ordinary positive subgoals of a $CQ^\neq^\neg q$
q^{cn}	arithmetic comparisons and negated subgoals of a $CQ^\neq^\neg q$
q^c	arithmetic comparisons of a $CQ^\neq^\neg q$
q^n	negated subgoals of a $CQ^\neq^\neg q$
$CMs(q_2^+, q_1^+)$	all the containment mappings from q_2^+ to q_1^+
$CMs(q^+, D)$	all the containment mappings from q^+ to D

shared variable X in q^+ except the first occurrence, replace the occurrence of X by a new distinct variable X_i , and add $X = X_i$ to q^c ; (2) For each constant c in the query, replace the constant by a new distinct variable Z , and add $Z = c$ to q^c .

THEOREM 2.7. [22] *Let q_1 and q_2 be the normalized CQ^\neq s. Let ρ_1, \dots, ρ_l be $CMs(q_2^+, q_1^+)$. Then $q_1 \sqsubseteq q_2$ if and only if $q_1^c \Rightarrow \rho_1(q_2^c) \vee \dots \vee \rho_l(q_2^c)$.*

In the rest of the paper, we distinguish the containment (\sqsubseteq) from the syntactical containment (\subseteq). We denote \subseteq as a pure subset relation of the queries, i.e., $q_1 \subseteq q_2$ means every element in q_1 exists in q_2 . The following trivial relationship holds:

LEMMA 2.8. *Let q_1 and q_2 be CQ^\neq^\neg s as follows:*

$$\begin{aligned} q_1 : \text{ans}() &\leftarrow q_1^+, q_1^{cn}. \\ q_2 : \text{ans}() &\leftarrow q_2^+, q_2^{cn}. \end{aligned}$$

where $q_2^+ \subseteq q_1^+$ and $q_2^{cn} \subseteq q_1^{cn}$. Then $q_1 \sqsubseteq q_2$.

2.2 Extensions of Formalisms for CQ^\neq^\neg s

2.2.1 Satisfiability and Compact Normal Form

It is well known that all CQ s are satisfiable. It can be easily shown that CQ^\neq is satisfiable if the conjunction of its arithmetic comparisons is satisfiable. For CQ^\neg s, the following holds:

COROLLARY 2.9 (FOLKLORE). *A CQ^\neg is satisfiable iff there does not exist a subgoal $g(\bar{X})$, such that $g(\bar{X}) \in q^+$ and $\neg g(\bar{X}) \in q^n$.*

However, satisfiability of a CQ^\neq^\neg can not be tested according to the syntax of the query. For instance, let q be the query shown in Introduction:

$$q : \text{ans}() \leftarrow p(X, Y), \neg p(Y, X), X = Y.$$

It is obvious that q is unsatisfiable although both subqueries $\text{ans}() \leftarrow p(X, Y), \neg p(Y, X)$. and $\text{ans}() \leftarrow p(X, Y), X = Y$. are satisfiable.

To overcome this problem, we introduce the concept of the *compact normal form*.

DEFINITION 2.10 (COMPACT NORMAL FORM). *A CQ^\neq^\neg q is in compact normal form if the arithmetic comparisons of q do not imply any equality with the form $X = Y$ or $X = a$.*

Any CQ^\neq^\neg can be transformed into its compact normal form as follows: (1) We first partition the variables and constants in q into n disjoint subsets (equivalence classes) based on their equivalence relations. Note that there is at most one constant in each class. (2) For all the elements in each equivalence class, choose one representative X_i (i.e., if the class contains a constant a , then $X_i = a$; otherwise X_i is assigned with one arbitrary variable). (3) Rewrite each subgoal and each arithmetic comparison by replacing every variable with its representative. Note that all the compact normal forms of a give query q are equivalent upon isomorphism.

The intuition behind the compact normal form is as follows: if a CQ^\neq^\neg is in compact normal form, then there is a total order of all variables so that a canonical database can be constructed by mapping each variable to a distinct constant.

PROPOSITION 2.11. *Let q be a CQ^\neq^\neg in compact normal form as follows:*

$$\text{ans}() \leftarrow p_1, \dots, p_n, cn_1, \dots, cn_m.$$

where for some $r \leq m$, every cn_i ($1 \leq i \leq r$) is an arithmetic comparison and every cn_i ($r+1 \leq i \leq m$) is a negated subgoal. Then q is satisfiable if and only if

1. the conjunction of cn_1, \dots, cn_r is satisfiable, and
2. there does not exist a subgoal $g_i(\bar{X}_i)$, such that $g_i(\bar{X}_i) \in \{p_1, \dots, p_n\}$ and $\neg g_i(\bar{X}_i) \in \{cn_{r+1}, \dots, cn_m\}$

PROOF. Since q is in compact normal form, it does not imply any equality with the form $X = Y$ or $X = a$, then according to the arithmetic comparisons cn_1, \dots, cn_r , we can construct a canonical database D as follows: generate a symbol mapping ρ that maps each constants in $\text{cons}(q^+)$ into the same constant and each variable in $\text{vars}(q^+)$ into a distinct new constant, such that $\rho(\text{vars}(q^+)) \cup \rho(\text{cons}(q^+))$ are in total order. Let $\text{adom}(D) = \rho(\text{vars}(q^+)) \cup \rho(\text{cons}(q^+))$, and $\rho(q^+)$ be the canonical database D . Note that ρ is a one-to-one mapping and for every tuple $t \in D$, $\rho^{-1}(t) \in q^+$ holds. Since the conjunction of cn_1, \dots, cn_r is satisfiable, then $\rho(cn_i)_{(1 \leq i \leq r)} = \text{TRUE}$ holds.

It remains to show that for every $i_{(r+1 \leq i \leq m)}$, $\rho(\text{co}(cn_i)) \notin D$. Suppose on the contrary there is one $cn_{i_{(r+1 \leq i \leq m)}}$, such that $\rho(\text{co}(cn_i)) \in D$. Because ρ is a one-to-one mapping, $\rho^{-1}(\rho(\text{co}(cn_i))) \in q^+$ holds, which implies that $\text{co}(cn_i) \in q^+$. This constructs a contradiction to the condition (2).

Since we have shown that $q(D) = \text{TRUE}$, q is thus satisfiable. \square

2.2.2 Core and Complete Form

The concept of the *compact normal form* introduced in Section 2.2.1 is crucial to the satisfiability test of $CQ^{\neq s}$. Moreover, we have shown that if a query is in compact normal form, a canonical database can be constructed by mapping the variables into distinct constants. This is an important property in that it enables a *inverse containment mapping* from the canonical database into the positive subgoals of the query.

However, given a query q , it is often desirable to obtain an equivalent query of q , without removing the subgoals, while preserving the property of compact normal form. For this purpose, we introduce the concept of *core* and *complete form*.

DEFINITION 2.12 (CORE). *Let q be a $CQ^{\neq s}$, and q' a compact normal form of q . All the subgoals (both positive and negated) of q' is the core of q .*

It should be pointed out that the definition of core here differs from the concept of core in the data exchange context. The query in compact normal form (i.e., containing only a core as subgoals) need not to be minimal in terms of the size of the subgoals. For instance, consider the following $CQ^{\neq s}$:

EXAMPLE 2.13.

$$q : \text{ans}() \leftarrow p(X, Y), p(X, X), X < 5.$$

Query q is not minimal, since the subgoal $p(X, Y)$ is superfluous. However, it is in compact normal form, since $X = Y$ is not implied. Thus $\{p(X, Y), p(X, X)\}$ is a core of q .

DEFINITION 2.14 (COMPLETE FORM). *A $CQ^{\neq s}$ q is complete if it contains a core of q .*

The transformation from a query into its complete form is straightforward. We first reduce the query q into its compact normal form q' , then add all the subgoals of q' into q .

PROPOSITION 2.15. *Let q be a $CQ^{\neq s}$, and q' is a complete form of q . $q \equiv q'$ holds.*

PROOF. Since $q \subseteq q'$, $q' \sqsubseteq q$ holds. It remains to prove that $q \sqsubseteq q'$. Let D be any database s.t. $q(D) = \text{TRUE}$, we show that $q'(D) = \text{TRUE}$. Let ρ be the containment mapping from q^+ to D . Let p be a positive subgoal such that $p = g(X_1, \dots, X_n)$, if $p \in q^+$ and $p \notin q^+$, then $g(X_1, \dots, X_n)$ is a core subgoal. We have to show that $\rho(g(X_1, \dots, X_n)) \in D$.

From the definition of compact normal form, p has a *source subgoal* $g(Y_1, \dots, Y_n)$ in q^+ , s.t. $X_i = Y_i$ ($1 \leq i \leq n$) holds. We thus have $\rho(g(Y_1, \dots, Y_n)) \in D$. Since ρ maps the equivalent variables into the same constant in D , we thus have $\rho(g(X_1, \dots, X_n)) = \rho(g(Y_1, \dots, Y_n))$, therefore $\rho(g(X_1, \dots, X_n)) \in D$.

Analogously, it is easy to show that for each negated core subgoal $cn \in q'$, $\rho(cn) \notin D$. We thus have shown that $q'(D) = \text{TRUE}$. \square

3. THEOREM AND ALGORITHMS FOR CONTAINMENT CHECKING

In this section, we concentrate on the containment checking problem on $CQ^{\neq s}$. Section 3.1 introduces the main theorem. In Section 3.2 we propose the basic algorithm **CONTAINMENT**, which is followed by the sub-function **APRIORI** discussed in Section 3.3.

3.1 Containment Theorem

THEOREM 3.1. *Let q_1 and q_2 be $CQ^{\neq s}$ as follows:*

$$\begin{aligned} q_1 : \text{ans}() &\leftarrow q_1^+, q_1^{cn}. \\ q_2 : \text{ans}() &\leftarrow q_2^+, q_2^{cn}. \end{aligned}$$

Then $q_1 \not\sqsubseteq q_2$ if and only if there exist a set of symbol mappings ρ_1, \dots, ρ_n from q_2 to q_1 and a satisfiable, complete query q of the form

$$q : \text{ans}() \leftarrow q^+, q^{cn}.$$

such that

1. $q_1^+ \subseteq q^+$ and $q_1^{cn} \subseteq q^{cn}$,
2. $q_2^{cn} = cn_1, \dots, cn_m$ and $\{\rho_1(\text{co}(cn_{l_1})), \dots, \rho_n(\text{co}(cn_{l_n}))\} \subseteq q^+ \cup q^{cn}$ where $\{l_1, \dots, l_n\} \subseteq \{1, \dots, m\}$, and
3. $\text{CMs}(q_2^+, q^+) \subseteq \{\rho_1, \dots, \rho_n\}$

PROOF. (1) \Rightarrow : Assume that there exists a satisfiable, complete query q , we show that there exists a database D , such that $q_1(D)$ is **TRUE** and $q_2(D)$ is **FALSE**.

Since q is complete, it contains a core. Let $\text{core}^+(q)$ ($\text{core}^-(q)$, resp.) be all the positive (negated, resp.) subgoals occurring in the core of q . Since query q is satisfiable, we can derive a total order for the variables and constants in $\text{core}^+(q)$. According to this total order, we *freeze* $\text{core}^+(q)$ into a database D by replacing each variable in $\text{core}^+(q)$ into a distinct constant, while each constant in $\text{core}^+(q)$ remains the same. We denote the containment mapping from $\text{core}^+(q)$ into D as τ . Note that the mapping τ is one-to-one. Furthermore, for each negated subgoal $cn \in \text{core}^-(q)$, $\tau(\text{co}(cn)) \notin D$, because q is satisfiable.

Note that the source of τ contains only those variables occurring in the core of q . Next we extend τ to γ , where the source of γ contains all the variables in q^+ . Let X be a variable which occurs in q^+ , but does not occur in $\text{core}^+(q)$, and Y be the representative variable (or constant) of X , which occurs in $\text{core}^+(q)$. If $Y \rightarrow c \in \tau$, then add $X \rightarrow c$ into γ . We show that γ is a containment mapping from q^+ to D : for every subgoal $g(X_1, \dots, X_n) \in q^+$, there is a corresponding subgoal $g(Y_1, \dots, Y_n) \in \text{core}^+(q)$, s.t. $X_i = Y_i$ ($1 \leq i \leq n$) hold. Thus $\gamma(g(X_1, \dots, X_n)) = \tau(g(Y_1, \dots, Y_n)) \in D$.

We have shown that γ is a containment mapping from q^+ to D . Analogously, it can be easily proved that for every $cn \in q^{cn}$, $\gamma(\text{co}(cn)) \notin D$. Hence we can conclude that $q(D) = \text{TRUE}$ holds. Because $q \sqsubseteq q_1$, we have $q_1(D) = \text{TRUE}$.

Now we prove that $q_2(D) = \text{FALSE}$. Let ρ be any containment mapping from q_2^+ to D . Since τ is a one-to-one mapping, $\rho \circ \tau^{-1}$ is a containment mapping from q_2^+ to q^+ . From condition (3) we know that $\rho \in \{\rho_1, \dots, \rho_n\}$, further due to condition (2), there exists a $cn \in q_2^{cn}$, such that $\rho \circ \tau^{-1}(\text{co}(cn)) \in q^+ \cup q^{cn}$. From the construction of τ , we know that if cn is an arithmetic comparison, then $\rho \circ \tau^{-1} \circ \tau(\text{co}(cn))$ is **TRUE**, thus $\rho(\text{co}(cn))$ is **TRUE**. Otherwise, cn is a negated subgoal, then we have $\rho \circ \tau^{-1}(\text{co}(cn)) \in q^+$, thus $\rho \circ \tau^{-1} \circ \tau(\text{co}(cn)) \in D$, then we have $\rho(\text{co}(cn)) \in D$. Therefore we can conclude that $q_2(D)$ is **FALSE**.

(2) \Leftarrow : We show that if $q_1 \not\sqsubseteq q_2$, then there exists a query q , that fulfills all the conditions in Theorem 3.1. Let D be a database, such that $q_1(D) = \text{TRUE}$, and $q_2(D) = \text{FALSE}$, q can be constructed incrementally as follows: Let γ be the containment mappings from q_1^+ to D , s.t. $q_1(D)$ is **TRUE**, and μ_1, \dots, μ_k be all the containment mappings from q_2^+

to q_1^+ . We start from μ_1 . Clearly, $\mu_1 \circ \gamma$ is a containment mapping from q_2^+ to D . Since $q_2(D)$ is FALSE, there are two possibilities: (1) there exists an arithmetic comparison $cn \in q_2^{cn}$, such that $\mu_1 \circ \gamma(\text{co}(cn))$ is TRUE, or (2) there exists a negated subgoal $cn \in q_2^{cn}$, such that $\mu_1 \circ \gamma(\text{co}(cn)) \in D$. For any of the two cases, we can add $\mu_1(\text{co}(cn))$ into the body of q_1 . Let this new query be q_1' . Clearly, q_1' is still satisfiable. Furthermore, it holds that γ is still the containment mapping from q_1' to D , such that $q_1'(D) = \text{TRUE}$. After proceeding the procedure k times, we obtain a new satisfiable query as follows:

$$q_1' : \text{ans}() \leftarrow q_1^+, q_1^{cn}, \mu_1(\text{co}(cn_{i_1})), \dots, \mu_k(\text{co}(cn_{i_k})).$$

Note that q_1' contains new non-negated subgoals and arithmetic comparisons, but no new variable is introduced. If q_1' is not complete, then we transform it into complete form. We denote this new complete query as q_1'' . From Proposition 2.15 we know that q_1'' is satisfiable as well, and more importantly, it still holds that γ is the containment mapping from q_1'' to D , such that $q_1''(D) = \text{TRUE}$.

Because of the introduction of new non-negated subgoals, it is possible that new containment mappings from q_2^+ to $q_1''^+$ appear. If there is no new mapping introduced, we stop and q_1'' is the required query. Otherwise, we repeat the same procedure. Since there are finite numbers of variables in q_1 and q_2 , and during the process no new variables are introduced, hence the number of the containment mappings can not exceed the number of symbol mappings from q_2 to q_1 , which is finite. Therefore the process either halts after q is found, or reaches the fixpoint, which has the following form:

$$q : \text{ans}() \leftarrow q_1^+, q_1^{cn}, \mu_1(\text{co}(cn_{i_1})), \dots, \mu_m(\text{co}(cn_{i_m})).$$

where μ_1, \dots, μ_m are all the symbol mappings from q_2 to q_1 . Hence q is satisfiable and fulfills all the conditions in Theorem 3.1. \square

From Theorem 3.1 a Π_2^P upper bound for containment checking of CQ^{\neq} queries can be easily obtained as follows: (1) guess a set of symbol mappings ρ_1, \dots, ρ_n from q_2 to q_1 , and a set of $cn_{i_1}, \dots, cn_{i_n}$ from q_2^{cn} ; (2) then construct the query q and transform it into complete form; (3) check whether all the containment mappings from q_2^+ to q^+ are in ρ_1, \dots, ρ_n . Since step (3) is an NP oracle, the Π_2^P upper bound is obvious. However, the number of possible sets of symbol mappings is prohibitively large (double exponential), a naive guess-and-test algorithm does not have any practical use.

3.2 The Containment Algorithm

Inspired from the \Leftarrow proof of Theorem 3.1, we need only to recursively check those containment mappings from q_2^+ to q_1^+ , instead of checking all the symbol mappings from q_2 to q_1 . Assume there are n containment mappings from q_2^+ to q_1^+ , and there are m elements in q_2^{cn} , then the number of all the extended queries is m^n . In general n is much smaller than all the symbol mappings, thus the search space is reduced.

The main algorithm CONTAINMENT is given in Algorithm 1. It takes q_1, q_2 and ρ_1, \dots, ρ_n as input parameters, where ρ_1, \dots, ρ_n are all the containment mappings from q_2^+ to q_1^+ . For the time being let us assume that the sub-function APRIORI returns all the satisfiable queries that are extended from q_1 by adding the subgoals or arithmetic comparisons with the form $\rho_i(\text{co}(cn_{i_i}))$. The technical details concerning

pruning strategies for APRIORI are discussed in the following section.

If APRIORI returns an emptyset, this means all the queries extended from q_1 with ρ_1, \dots, ρ_n are unsatisfiable. Thus every possible query q which fulfills the conditions in Theorem 3.1 is unsatisfiable. CONTAINMENT returns YES. Otherwise, we transform all the extended queries into their complete form (line 4). Among all these complete queries, if there is a satisfiable query q such that no new containment mapping from q_2^+ to q^+ is generated, the algorithm halts with the answer NO. Otherwise, we continue the checking process with these extended queries, but with only the newly generated containment mappings considered. Since the number of symbol mappings is finite, the algorithm will finally reach the fixpoint and halt.

Algorithm 1 CONTAINMENT($q_1, q_2, \bar{\rho}$)

Input: $\bar{\rho} = \rho_1, \dots, \rho_n$ are all the containment mappings from q_2^+ to q_1^+ .

Output: YES if $q_1 \sqsubseteq q_2$, NO otherwise.

```

1: if  $\{q_1', \dots, q_k'\} = \text{APRIORI}(q_1, q_2, \rho_1, \dots, \rho_n)$  is  $\emptyset$  then
2:   return YES
3: end if
4: Transform every  $q_i'$  into complete form
5: for  $i = 1$  to  $k$  do
6:    $\bar{\gamma}_i = \text{newmappings}(q_i', q_2, \bar{\rho})$ 
7:   if  $\bar{\gamma}_i == \emptyset$  then
8:     return NO
9:   end if
10: end for
11: for  $i = 1$  to  $k$  do
12:   ans = CONTAINMENT( $q_i', q_2, \bar{\gamma}_i$ )
13:   if ans == NO then
14:     return NO
15:   end if
16: end for
17: return YES

```

Before we come to the sub-function APRIORI, we prove that the algorithm CONTAINMENT is correct and complete.

THEOREM 3.2. *Algorithm CONTAINMENT is correct and complete.*

PROOF. the correctness proof is trivial. In the following we prove the completeness. Let q_1 and q_2 be CQ^{\neq} s. Assume that there exists a set of symbol mappings $\{\rho_1, \dots, \rho_n\}$ and a satisfiable and complete query q as in Theorem 3.1. We show that algorithm CONTAINMENT returns NO.

Let ρ_1, \dots, ρ_r be all the containment mappings from q_2^+ to q_1^+ . Since $q_1^+ \subseteq q^+$, we have $\{\rho_1, \dots, \rho_r\} \subseteq \text{CMS}(q_2^+, q^+)$, thus $\{\rho_1, \dots, \rho_r\} \subseteq \{\rho_1, \dots, \rho_n\}$ holds. Then query q_1' can be constructed as follows:

$$q_1' : \text{ans}() \leftarrow q_1, \rho_1(\text{co}(cn_{i_1})), \dots, \rho_r(\text{co}(cn_{i_r})).$$

where $\{\rho_1(\text{co}(cn_{i_1})), \dots, \rho_r(\text{co}(cn_{i_r}))\} \subseteq q$. Since $q_1' \subseteq q$ holds, q_1' is satisfiable. The sub-function APRIORI (line 1) returns all the satisfiable queries extended from q_1 with ρ_1, \dots, ρ_r . Thus q_1' must be in the return set of APRIORI.

We then transform q_1' into its complete form. Note that $q_1' \subseteq q$ still holds. If no more new containment mappings from q_2^+ to $q_1'^+$ is generated, CONTAINMENT algorithm returns NO (line 8). Otherwise the function is recursively called with the new containment mappings. A sequence of

satisfiable queries is generated with the relationship:

$$q_1 \subseteq q_1' \subseteq \dots \subseteq q$$

by adding consecutively positive subgoals or arithmetic comparisons with the form $\rho_i(\text{co}(cn_{i_i}))$ where $1 \leq i \leq n$. Since CONTAINMENT deploys an exhaustive search, the fixpoint will be reached as soon as q (or a subset of it) is generated. Algorithm CONTAINMENT thus returns NO as well (line 14). \square

3.3 The Apriori Algorithm

In Section 3.2 we have presented the basic part of the containment checking algorithm. Given the upper bound, which is decided by the queries, algorithm CONTAINMENT could hardly be improved further. Now we consider the subfunction APRIORI. Each time the algorithm CONTAINMENT calls APRIORI, a set of satisfiable queries extended from q_1 is returned. Note that the number of queries is possibly exponential in the size of the number of containment mappings. Therefore the operation in APRIORI is costly on both time and space, and it is crucial to explore the opportunities to optimize the APRIORI algorithm.

Note that although an exponential number of extensions are to be tested, only the satisfiable queries are returned, so we are interested in the question: *how to detect those unsatisfiable queries as early as possible?* Indeed, if a query q is unsatisfiable, then any extended queries by adding new subgoals into q is unsatisfiable as well. Thus any further extension of q can be pruned. We formulate this property with the following corollary:

COROLLARY 3.3. *Given two queries q_1 and q_2 as follows:*

$$\begin{aligned} q_1 &: \text{ans}() \leftarrow q_1^+, q_1^{cn}. \\ q_2 &: \text{ans}() \leftarrow q_2^+, q_2^{cn}. \end{aligned}$$

where $q_1^+ \subseteq q_2^+$ and $q_1^{cn} \subseteq q_2^{cn}$. If q_1 is unsatisfiable, then q_2 is unsatisfiable.

The property of Corollary 3.3 is called *anti-monotonicity*, which is fundamental to designing the Apriori-like algorithm with desirable pruning strategies [3]. Simply stated, with the Apriori algorithm the query q_1 is extended level wise. At each level i , the query is extended only if all the queries that are more general at level $i-1$ are satisfiable. Otherwise the query will not be further extended.

Let q_1 and q_2 be CQ^{\neq} s and ρ_1, \dots, ρ_n be all the containment mappings and $q_2^{cn} = cn_1, \dots, cn_m$. The level wise algorithm APRIORI (Algorithm 2) proceeds iteratively as follows: At level one, we generated n unary relations $R_1(\rho_1), \dots, R_n(\rho_n)$. Each relation R_i contains m tuples with the form $\rho_i(\text{co}(cn_j))(1 \leq j \leq m)$. We remove those tuples t where $q_1 \wedge t$ is unsatisfiable. In each subsequent iteration, at level $i+1$ all the $i+1$ -ary tuples are generated from level i . For each $i+1$ -tuple t , if any i -ary projection of t does not exist in the relations at level i , t will be deleted. This is realized with the semi-join operation at line 16 to 22 in Algorithm 2.

Early termination. During the iteration, if any relation R becomes empty, the algorithm halts and returns the empty set (line 11). This is easy to understand, because every possible tuple in the relation at the last level contains at least one tuple in R .

At the last level of the iteration, we obtain a single relation in which every tuple t has the property that $q_1 \wedge t$ is satisfiable. These tuples are returned for further processing in Algorithm 1.

Algorithm 2 APRIORI($q_1, q_2, \bar{\rho}$)

Input: $\bar{\rho} = \rho_1, \dots, \rho_n$ are the containment mappings from q_2^+ to q_1^+ , $q_2^{cn} = cn_1, \dots, cn_m$.

Output: A set of satisfiable queries more specific than q_1 with the form $q_1 \wedge \bigwedge_{i=1}^n \rho_i(\text{co}(cn_{i_i}))$

```

1: Generate  $n$  one-column relations  $L_1 = \{R_1, \dots, R_n\}$ 
   where the attribute for  $R_i$  is  $\rho_i$ 
2: for  $i = 1$  to  $n$  do
3:   for  $j = 1$  to  $m$  do
4:     Insert tuple  $\rho_i(\text{co}(cn_j))$  into  $R_i$ 
5:   end for
6: end for
7:  $k = 1$ 
8: repeat
9:   Delete every tuple  $t$  in relations of  $L_k$  where  $q_1 \wedge t$  is
   unsatisfiable
10:  if any  $k$ -column relation in  $L_k$  is empty then
11:    return  $\emptyset$ 
12:  end if
13:  Generate all the  $k+1$ -column relations  $L_{k+1}$  by the
   natural-join of relations in  $L_k$ 
14:  for all relation  $R$  in  $L_{k+1}$  do
15:    for all relation  $R'$  in  $L_k$  do
16:      if  $\text{Attr}(R') \subseteq \text{Attr}(R)$  then
17:         $R = R \times R'$ 
18:      end if
19:    end for
20:  end for
21:   $k = k + 1$ 
22: until  $k > n$ 
23: Let  $L_n = \{R\}$  /* there is one table in  $L_n$  */
24: Delete every tuple  $t$  in  $R$  where  $q_1 \wedge t$  is unsatisfiable
25: Add  $q_1$  to every tuple in  $R$ 
26: return  $R$ 

```

We denote the relation at the last level as *final relation* and the tuples in the final relation as *final tuples*. To simplify the presentation, given a relation R generated in the algorithm APRIORI and a tuple t in R , if the context is clear, we use *the tuple t is satisfiable* by meaning that *the query $q_1 \wedge t$ is satisfiable*.

With the following Example 3.4, which is adapted from [2], we demonstrate the containment checking process following CONTAINMENT and APRIORI algorithms. Note that in [2] the authors used this example by showing that the *normalization* step of ZOG algorithm can not be avoided. We will illustrate that the normalization step is done here on the fly.

EXAMPLE 3.4. [2] *Consider the following queries q_1 and q_2 :*

$$\begin{aligned} q_1 &: \text{ans}() \leftarrow p(X, Y, 2, 1, U, U), p(1, 2, X, Y, U, U), \\ & \quad p(1, 2, 2, 1, X, Y). \\ q_2 &: \text{ans}() \leftarrow p(X, Y, Z, Z', U, U), X < Y, Z > Z'. \end{aligned}$$

We check whether $q_1 \sqsubseteq q_2$. According to the algorithm CONTAINMENT, we first generate all the containment mappings from q_2^+ to q_1^+ as follows:

$$\begin{aligned} \rho_1 &: X \rightarrow X, Y \rightarrow Y, Z \rightarrow 2, Z' \rightarrow 1, U \rightarrow U \\ \rho_2 &: X \rightarrow 1, Y \rightarrow 2, Z \rightarrow X, Z' \rightarrow Y, U \rightarrow U \end{aligned}$$

The APRIORI algorithm is then called. At level 1, we generate two relations for ρ_1 and ρ_2 which are illustrated in Table 2 (a) and (b). In both relations, we remove the tuples which

result in unsatisfiability. Thus tuple $2 \leq 1$ and $1 \geq 2$ are removed. As a result, both relations contain one tuple. At level 2, all relations with two attributes are generated. In this case, one relation is generated, which is shown in Table 2 (c). Because both sub-relations of the tuple R_3 exist at level 1, R_3 succeeds the semi-join test.

A new query q is returned from APRIORI, with the arithmetic comparisons in R_3 ($X \leq Y$ and $X \geq Y$) added:

$$q: \text{ans}() \leftarrow p(X, Y, 2, 1, U, U), p(1, 2, X, Y, U, U), \\ p(1, 2, 2, 1, X, Y), X \leq Y, X \geq Y.$$

Since q implies $X = Y$, we transform q into its complete form q' :

$$q': \text{ans}() \leftarrow p(X, Y, 2, 1, U, U), p(1, 2, X, Y, U, U), \\ p(1, 2, 2, 1, X, Y), X \leq Y, X \geq Y. \\ \underline{p(X, X, 2, 1, U, U)}, \underline{p(1, 2, X, X, U, U)}, \\ \underline{p(1, 2, 2, 1, X, X)}.$$

where the underlined subgoals are newly generated due to the equivalence $X = Y$. Three new containment mappings are thus generated:

$$\rho_3: X \rightarrow X, Y \rightarrow X, Z \rightarrow 2, Z' \rightarrow 1, U \rightarrow U \\ \rho_4: X \rightarrow 1, Y \rightarrow 2, Z \rightarrow X, Z' \rightarrow X, U \rightarrow U \\ \rho_5: X \rightarrow 1, Y \rightarrow 2, Z \rightarrow 2, Z' \rightarrow 1, U \rightarrow X$$

and the APRIORI procedure is called, however with newly generated containment mappings ρ_3 , ρ_4 and ρ_5 . The same procedure is executed and three relations are generated at level 1. as illustrated in Table 2 (d), (e) and (f). Since every tuple in R_6 results in a unsatisfiable query, the early termination condition is fulfilled. Thus the APRIORI sub-function returns \emptyset . The algorithm CONTAINMENT finally returns YES.

Table 2: Relations generated for Example 3.4

(a) R_1	(b) R_2	First call level 1	
ρ_1	ρ_2		
$X \geq Y$	$1 \geq 2$		
$2 \leq 1$	$X \leq Y$		
(c) R_3		First call level 2	
ρ_1	ρ_2		
$X \geq Y$	$X \leq Y$		
(d) R_4	(e) R_5	(f) R_6	Second call level 1
ρ_3	ρ_4	ρ_5	
$X \geq X$	$1 \geq 2$	$1 \geq 2$	
$2 \leq 1$	$X \leq X$	$2 \leq 1$	

4. FURTHER IMPROVEMENTS

In this section, we propose two algorithms to further improve the time and space efficiency of the algorithm APRIORI, with the following objectives: (1) minimization of the number of the final tuples, and (2) minimization of the intermediate tests in the level wise iteration.

Section 4.1 considers the improvement of the algorithm in terms of task (2), and in Section 4.2 we propose further improvements concerning task (1).

4.1 Node Clustering

Although our algorithm makes use of the concept of the Apriori algorithm to process the satisfiability checking, it differs from the classical problem setting in that our algorithm returns only the final tuples at the last step L_n . The generation of the intermediate tuples can prevent the unsatisfiable tuples from further being extended at an early stage, but these intermediate tuples are not returned. Therefore if the number of the intermediate tuples can be reduced, the time and space consumption of the algorithm will be accordingly reduced.

Assume that at level i of the APRIORI algorithm, if several relations contain only one tuple, the conjunction of all these tuples will occur in every tuple of the final relation. Thus if all these tuples are *clustered* into one relation, many intermediate relations containing subsets of these tuples need not to be generated. We denote this operation as *node clustering*.

Procedure 3 Node Clustering

- 1: Let R_1, \dots, R_r be all the single tuple relations in L_k
 - 2: $R = R_1 \bowtie \dots \bowtie R_r$
 - 3: Let R_{r+1}, \dots, R_m be all the relations such that $\text{Attr}(R_i) \cap \text{Attr}(R) \neq \emptyset$ where $(r+1 \leq i \leq m)$
 - 4: **for** $i = r+1$ to m **do**
 - 5: $R = R \bowtie R_i$
 - 6: **end for**
 - 7: remove R_1, \dots, R_m from L_k , add R into L_k
 - 8: $k = |R|$
-

We give a sketch of the *node clustering* algorithm in Procedure 3. This procedure can be inserted between the line 12 and 13 of the APRIORI algorithm. Note that in the last line of Procedure 3, the level number k is increased to $|R|$, so that the intermediate relations between level $|R|$ and k need not to be generated.

We illustrate the node clustering operation with Example 4.1.

EXAMPLE 4.1. Consider the conjunctive queries q_1 and q_2 as follows:

$$q_1: \text{ans}() \leftarrow p(X, Y, Z), p(Z, U, V), \neg p(X, Y, U). \\ q_2: \text{ans}() \leftarrow p(A, B, C), p(D, E, F), B \neq F, D \neq E.$$

The containment mappings from q_2^+ to q_1^+ are as follows:

$$\rho_1: A \rightarrow X, B \rightarrow Y, C \rightarrow Z, D \rightarrow Z, E \rightarrow U, F \rightarrow V \\ \rho_2: A \rightarrow X, B \rightarrow Y, C \rightarrow Z, D \rightarrow X, E \rightarrow Y, F \rightarrow Z \\ \rho_3: A \rightarrow Z, B \rightarrow U, C \rightarrow V, D \rightarrow Z, E \rightarrow U, F \rightarrow V \\ \rho_4: A \rightarrow Z, B \rightarrow U, C \rightarrow V, D \rightarrow X, E \rightarrow Y, F \rightarrow Z$$

The relations generated at level 1 are given in Table 3. After the satisfiability checking of the first level, the tuple $Z = U$ is removed from R_1 , R_3 and R_4 because $q_1 \wedge (Z = U)$ is unsatisfiable. Now each of the relations R_1 , R_3 and R_4 contains one tuple, as shown in (e), (g) and (h) of Table 3. We then cluster these relations into a new relation R_5 , which is illustrated in (i) of Table 3.

Since the correctness of the node clustering operation is straightforward, we omit the proof here.

4.2 Tuple Pruning

When considering a tuple t in the final relation, we have so far made the assumption that the satisfiability of a tuple is independent from other tuples. In this section, we analyze

Table 3: Relations generated for Example 4.1. Relations (a),(b),(c),(d) are generated at level 1, with relations (e),(f),(g),(h) after the deletion of the tuple $Z = U$, and the clustered relations (i),(j) after node clustering

(a) R_1	(b) R_2	(c) R_3	(d) R_4
ρ_1	ρ_2	ρ_3	ρ_4
$Y = V$	$Y = Z$	$U = V$	$Z = U$
$Z = U$	$X = Y$	$Z = U$	$X = Y$
(e) R_1	(f) R_2	(g) R_3	(h) R_4
ρ_1	ρ_2	ρ_3	ρ_4
$Y = V$	$Y = Z$	$U = V$	$X = Y$
	$X = Y$		
(i) R_5			(j) R_2
ρ_1	ρ_3	ρ_4	ρ_2
$Y = V$	$U = V$	$X = Y$	$Y = Z$
			$X = Y$

the *inter-tuple* relationship, and show that a tuple t can become redundant, due to the existence of another tuple t' as its *witness*, although t itself is satisfiable.

Let us consider a simple example. Assume that at level i , there is a tuple t containing $X < 15$, and at level $i + 1$, we extend t to t_1 and t_2 with the attribute ρ by adding $\rho(\text{co}(cn_1))$ and $\rho(\text{co}(cn_2))$ respectively. Assume $\rho(\text{co}(cn_1)) = X < 16$ and $\rho(\text{co}(cn_2)) = X > 8$. Clearly, $t \rightarrow X < 16$ holds. This means, adding $\rho(\text{co}(cn_1))$ to t does not make t more specific (i.e., t_1 is equivalent to t). We call t_1 is a *witness* of t_2 and consider t_2 as a redundant tuple which can be removed, although t_2 itself is satisfiable. We name such an operation as *tuple pruning*.

The intuition behind tuple pruning is as follows: if at last a satisfiable query q is found, which contains t_2 and fulfills all the conditions in Theorem 3.1, then we can prove that another query q' exists, by simply adding $\rho(\text{co}(cn_1))$ into q , such that q' fulfills the conditions in Theorem 3.1 as well. In Procedure 4 we give the sketch of the *tuple pruning* algorithm. This procedure can be inserted between line 12 and 13 of Algorithm 2, however before the procedure *node clustering*.

Note that implication (\rightarrow) in Procedure 4 can be defined as follows: let q be a $CQ^{\neg\neq}$, ρ be some containment mapping and cn be either an arithmetic comparison or a negated subgoal, then $q \rightarrow \rho(\text{co}(cn))$ means that $q \wedge \rho(\text{co}(cn))$ is unsatisfiable.

Procedure 4 Tuple Pruning

Let $R = (\rho_1, \dots, \rho_k)$ be a k -ary relation in L_k
if there is a tuple $\langle \rho_1(\text{co}(cn_{l_1})), \dots, \rho_k(\text{co}(cn_{l_k})) \rangle$ in R s.t.
 $q_1, \rho_1(\text{co}(cn_{l_1})), \dots, \rho_{i-1}(\text{co}(cn_{l_{i-1}})), \rho_{i+1}(\text{co}(cn_{l_{i+1}})), \dots,$
 $\rho_k(\text{co}(cn_{l_k})) \rightarrow \rho_i(\text{co}(cn_{l_i}))$
then Delete the tuples in R with the form
 $\langle \rho_1(\text{co}(cn_{l_1})), \dots, \rho_{i-1}(\text{co}(cn_{l_{i-1}})), \rho_i(\text{co}(cn_{l_j})),$
 $\rho_{i+1}(\text{co}(cn_{l_{i+1}})), \dots, \rho_k(\text{co}(cn_{l_k})) \rangle$
 where $l_i \neq l_j$
endif

THEOREM 4.2. *Procedure 4 is correct.*

PROOF. We show that during the iteration, if any tuple with the form $t = \langle \rho_1(\text{co}(cn_{l_1})), \dots, \rho_{i-1}(\text{co}(cn_{l_{i-1}})), \rho_i(\text{co}(cn_{l_j})), \rho_{i+1}(\text{co}(cn_{l_{i+1}})), \dots, \rho_k(\text{co}(cn_{l_k})) \rangle$ is deleted according to Procedure 4, due to the existence of the witness tuple $t' = \langle \rho_1(\text{co}(cn_{l_1})), \dots, \rho_{i-1}(\text{co}(cn_{l_{i-1}})), \rho_i(\text{co}(cn_{l_i})), \rho_{i+1}(\text{co}(cn_{l_{i+1}})), \dots, \rho_k(\text{co}(cn_{l_k})) \rangle$, then if there is any satisfiable and complete query q and ρ_1, \dots, ρ_n as in Theorem 3.1, such that the conditions are fulfilled, and q contains t , then there exists a corresponding query q' , which is satisfiable and complete, and fulfills the conditions as well.

q' can be obtained by simply adding $\rho_i(\text{co}(cn_{l_i}))$ into q . Clearly, q' is satisfiable and complete. Condition (1) and (2) of Theorem 3.1 holds for q' as well. Furthermore, since $CMS(q_2^+, q'^+) = CMS(q_2^+, q^+)$, we have $CMS(q_2^+, q'^+) \subseteq \{\rho_1, \dots, \rho_n\}$.

Since the CONTAINMENT algorithm is complete, it will return NO due to the existence of q' . \square

The pruning of redundant tuples is effective in that if a witness tuple is identified, several redundant tuples (which depend on the size of q_2^{cn}) can be removed. Moreover, if the pruning is executed at level 1 during the iteration of the APRIORI algorithm, we usually obtain single tuple relations, which can be further processed by using *node clustering*. In the following example, we show that by integrating both strategies, the containment checking can achieve a drastic speed up.

EXAMPLE 4.3. [20] *Given the queries q_1 and q_2 :*

$$q_1 : \text{ans}() \leftarrow p(X, Y), p(Y, Z), \neg p(X, Z).$$

$$q_2 : \text{ans}() \leftarrow p(A, B), p(C, D), \neg p(A, D), \neg p(B, C)$$

There are four containment mappings from q_2^+ to q_1^+ :
 $\rho_1: A \rightarrow X, B \rightarrow Y, C \rightarrow Y, D \rightarrow Z,$
 $\rho_2: A \rightarrow X, B \rightarrow Y, C \rightarrow X, D \rightarrow Y,$
 $\rho_3: A \rightarrow Y, B \rightarrow Z, C \rightarrow Y, D \rightarrow Z,$
 $\rho_4: A \rightarrow Y, B \rightarrow Z, C \rightarrow X, D \rightarrow Y.$

By calling the APRIORI algorithm, we obtain 4 relations: R_1, R_2, R_3 and R_4 at level 1, which are illustrated as (a)-(d) in Table 4. Note that tuple deletion applied to R_1 is due to the fact that $q_1 \wedge p(X, Z)$ is unsatisfiable. While for R_2 and R_3 , *tuple pruning* is executed. To see this, let us consider R_2 . $p(X, Y)$ is a positive subgoal in q_1 , thus $q_1 \rightarrow p(X, Y)$ holds. Therefore $p(X, Y)$ is a witness of $p(Y, X)$ and $p(Y, X)$ can be deleted.

Since there are three single tuple relations, *node clustering* is applied. Note that with *node clustering* we directly reach level 3, without generating any relations for level 2. At level 4, the final relation is generated as shown in (k) of Table 4. Again *tuple pruning* is applied and we can thus prune the second tuple. At last, the APRIORI algorithm returns only one final tuple, whereas without the deployment of *tuple pruning*, the final relation could have contained 8 tuples. Moreover, with *node clustering* we have achieved skipping one iteration level, thus the generation of many intermediate relations are omitted.

Next we continue with extending the query

$$q_1' : \text{ans}() \leftarrow p(X, Y), p(Y, Z), \neg p(X, Z), \underline{p(Y, Y)}.$$

with the newly added subgoal $p(Y, Y)$. We will leave the details out here. In fact, q_1' is the desired query which fulfills all the conditions in Theorem 3.1. Therefore we conclude that $q_1 \not\sqsubseteq q_2$.

Table 4: Relations generated for Example 4.3

(a) R_1	(b) R_2	(c) R_3	(d) R_4
ρ_1	ρ_2	ρ_3	ρ_4
$p(X, Z)$	$p(X, Y)$	$p(Y, Z)$	$p(Y, Y)$
$p(Y, Y)$	$p(Y, X)$	$p(Z, Y)$	$p(Z, X)$
level 1			
(e) R_1	(f) R_2	(g) R_3	(h) R_4
ρ_1	ρ_2	ρ_3	ρ_4
$p(Y, Y)$	$p(X, Y)$	$p(Y, Z)$	$p(Y, Y)$
			$p(Z, X)$
level 1 tuple pruning			
(i) R_5	(j) R_4		
ρ_1	ρ_2	ρ_3	ρ_1
$p(Y, Y)$	$p(X, Y)$	$p(Y, Z)$	$p(Y, Y)$
			$p(Z, X)$
level 3 node clustering			
(k) R_6			
ρ_1	ρ_2	ρ_3	ρ_4
$p(Y, Y)$	$p(X, Y)$	$p(Y, Z)$	$p(Y, Y)$
$p(Y, Y)$	$p(X, Y)$	$p(Y, Z)$	$p(Z, X)$
level 4			
(l) R_6			
ρ_1	ρ_2	ρ_3	ρ_4
$p(Y, Y)$	$p(X, Y)$	$p(Y, Z)$	$p(Y, Y)$
level 4 tuple pruning			

5. CONCLUSIONS

In this paper, we have proposed a new approach for solving the problem of containment checking for conjunctive queries with both arithmetic comparisons and safe negations. We have shown that in spite of the computational intractability of the problem, our Apriori-like algorithms achieved drastic reduction on time and space consumption. We also demonstrated that the usage of node clustering and tuple pruning techniques can further leverage the performance of the containment checking algorithm.

Different test cases demonstrated in this paper have shown that our method of searching for a practical solution of a hard problem by exploring algorithmic optimizations is promising. Indeed, there is still much space for further improvements. Many variations of the Apriori algorithm have been proposed that focus on improving the efficiency of the original algorithm, such as Hash-based technique [15], partitioning [17], and dynamic itemset counting approach [4]. We believe that our algorithms can be further optimized by exploring these approaches.

One of our future work is to test our query containment algorithm on benchmark queries. By analyzing the characteristics of queries from different applications, such as queries over decision support systems and OLTP queries, more spe-

cific strategies can thus be developed.

6. REFERENCES

- [1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley Publishing Company, 1995.
- [2] F. N. Afrati, C. Li, and P. Mitra. On containment of conjunctive queries with arithmetic comparisons. In *EDBT*, pages 459–476, 2004.
- [3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.
- [4] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD*, pages 255–264, 1997.
- [5] A. K. Chandra and P. M. Merlin. Optimal implementations of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.
- [6] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
- [7] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.
- [8] A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint checking with partial information. In *PODS*, pages 45–55, 1994.
- [9] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.
- [10] A. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, 1988.
- [11] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *PODS*, pages 205–213, 1998.
- [12] M. Leclère and M.-L. Mugnier. Some algorithmic improvements for the containment problem of conjunctive queries with negation. In *ICDT*, pages 404–418, 2007.
- [13] A. Y. Levy and Y. Sagiv. Queries independent of updates. In *VLDB*, pages 171–181, 1993.
- [14] M. Ortiz, D. Calvanese, and T. Eiter. Characterizing data complexity for conjunctive query answering in expressive description logics. In *AAAI*, 2006.
- [15] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash based algorithm for mining association rules. In *SIGMOD*, pages 175–186, 1995.
- [16] Y. Saraiya. *Subtree elimination algorithms in deductive databases*. PhD thesis, Stanford University, 1991.
- [17] A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. In *VLDB*, pages 432–444, 1995.
- [18] J. D. Ullman. Information integration using logical views. In *ICDT*, pages 19–40, 1997.
- [19] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *PODS*, pages 331–345, 1992.
- [20] F. Wei and G. Lausen. Containment of conjunctive queries with safe negation. In *ICDT*, pages 343–357, 2003.
- [21] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.
- [22] X. Zhang and Z. M. Özsoyoglu. Implication and referential constraints: A new formal reasoning. *TKDE*, 9(6):894–910, 1997.