Subsumption and indexing in constraint query languages with linear arithmetic constraints

Divesh Srivastava*

Computer Sciences Department, University of Wisconsin-Madison, WI 53706, USA

Abstract

Bottom-up evaluation of a program-query pair in a constraint query language (CQL) starts with the facts in the database and repeatedly applies the rules of the program, in iterations, to compute new facts, until we have reached a fixpoint. Checking if a fixpoint has been reached amounts to checking if any "new" facts were computed in an iteration. Such a check also enhances efficiency in that subsumed facts can be discarded, and not be used to make any further derivations in subsequent iterations, if we use Semi-naive evaluation. We show that the problem of subsumption in CQLs with linear arithmetic constraints is co-NP complete, and present a deterministic algorithm, based on the divide and conquer strategy, for this problem. We also identify polynomial-time sufficient conditions for subsumption and non-subsumption in CQLs with linear arithmetic constraints. We adapt indexing strategies from spatial databases for efficiently indexing facts in such a CQL: such indexing is crucial for performance in the presence of large databases. Based on a recent algorithm by C. Lassez and J.-L. Lassez for quantifier elimination, we present an incremental version of the algorithm to check for subsumption in CQLs with linear arithmetic constraints.

1. Introduction

Recently, there have been attempts ([2,4,9,13] among others) to increase the expressive power of database query languages by integrating constraint paradigms with logic-based database query languages; such languages are referred to as *constraint query languages* (CQLs). Constraint query languages retain the important declarative aspect of database query languages since constraint programming paradigms are inherently declarative. Bottom-up evaluation of a program in a CQL is very important since it is sound and complete with respect to the declarative semantics [7] of such programs. Bottom-up evaluation also offers considerable scope for optimization, which is essential since evaluating such programs can be expensive due to the manipulation of constraints.

* This work was supported by a David and Lucile Packard Foundation Fellowship in Science and Engineering, a Presidential Young Investigator Award, with matching grants from the Digital Equipment Corporation, Tandem and Xerox, and NSF Grant No. IRI-9011563.

Bottom-up evaluation of a program in a CQL starts with the facts in the database and repeatedly applies all the rules of the program, in iterations, to compute new facts. The evaluation terminates once we have reached a fixpoint. A fundamental aspect of bottom-up evaluation is that we must constantly check to see if the fixpoint has been reached. This amounts to checking if any new facts were computed in an iteration. Such a check also enhances efficiency in that subsumed facts can be discarded, and not be used to make any further derivations in subsequent iterations, if we use Semi-naive evaluation [1,3].

Facts in a constraint query language (referred to as *constraint facts*) are conjunctions of constraints. Relations are finite collections of facts, as is usual in database query languages. In this paper, we concern ourselves with the problem of subsumption in constraint query languages where the only constraints permitted are linear arithmetic constraints. Constraint facts in such a CQL can be viewed geometrically as convex polyhedra [14], and relations can be viewed geometrically as finite (non-convex) unions of convex polyhedra. Checking whether a newly computed constraint fact is subsumed by the existing constraint facts in a relation also has a geometric interpretation. It is the problem of checking whether a convex polyhedron is contained within a finite union of convex polyhedra.

Our contributions are as follows:

(1) We show that determining whether a convex polyhedron is contained in a finite union of convex polyhedra is co-NP complete (section 3).

(2) We present a deterministic algorithm, based on the divide and conquer strategy, to determine whether a convex polyhedron is contained in a finite union of convex polyhedra (section 4).

(3) We adapt indexing strategies from spatial databases for efficiently indexing convex polyhedra (section 5). Such indexing is crucial for performance in the presence of large databases of constraint facts.

(4) We identify polynomial time sufficient conditions to check when a convex polyhedron is or is not contained in a finite union of convex polyhedra (section 6).

(5) We present an incremental variant of the algorithm presented in section 4 to check whether a convex polyhedron is contained in a finite union of convex polyhedra (section 7).

The results are also of independent interest to researchers in linear programming.

2. Preliminaries

2.1. BOTTOM-UP EVALUATION OF CQL PROGRAMS

We assume familiarity with the syntax and semantics of constraint logic programs, as well as the issues involved in the bottom-up evaluation of such programs (see [7,9] for details). A few important definitions are given here:

DEFINITION 2.1: LINEAR ARITHMETIC CONSTRAINT

A *linear arithmetic constraint* is of the form:

$$a_1X_1 + \ldots + a_mX_m \quad op \quad a_{m+1},$$

where $a_1, \ldots, a_{m+1}$ are real-valued constants, and the operator *op* is one of $<$ or $\leq$.

Constraints involving $>, \geq$ and $=$ can be rewritten as conjunctions of constraints involving only $<$ and $\leq$, and we use such constraints in our examples.

DEFINITION 2.2: NEGATION OF A LINEAR ARITHMETIC CONSTRAINT

Given a constraint $c \equiv a_1X_1 + \ldots + a_mX_m \leq a_{m+1}$ (respectively, $a_1X_1 + \ldots + a_mX_m < a_{m+1}$), the *negation* of c, denoted by $\neg c$, is the linear arithmetic constraint $a_1X_1 + \ldots + a_mX_m > a_{m+1}$ (respectively, $a_1X_1 + \ldots + a_mX_m \geq a_{m+1}$).

A *rule* in a CQL is of the form:

$$p(\overline{X}) :- C, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}),$$

where C is a conjunction of constraints, and $p(\overline{X}), p_1(\overline{X_1}), \ldots, p_n(\overline{X_n})$ are atoms. $p(\overline{X})$ is referred to as the *head* of the rule, and $C, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n})$ is referred to as the *body* of the rule. A rule with no body atoms (although it could have a conjunction of constraints in the body) is referred to as a *constraint fact.* It is also represented as $p(\overline{X}; C)$, and is thus a conjunction of constraints [9,12]. It is a finite representation of the (potentially) infinite set of ground facts that satisfy the conjunction of constraints C.

A *relation* is a finite collection of such facts, i.e. a disjunction of conjunctions of constraints. A *database* is a finite set of relations. A *program* is a finite set of rules, and the meaning of a program is given by its least model [7].

Bottom-up evaluation of a program in a CQL proceeds by starting with the facts in the database and repeatedly applying all the rules of the program, in iterations, to compute new facts. The evaluation terminates once we have reached a fixpoint. We now intuitively describe a rule application, the basic step in bottom-up evaluation.

DEFINITION 2.3: RULE APPLICATION

Consider a program rule:

$$r : p(\overline{X}) :- C, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}),$$

where r is just a label we use, and is not part of the syntax of a rule. A *derivation* of a p fact using rule r consists of two steps:

- First, choose one p_i fact for each $p_i(\overline{X_i})$, $1 \leq i \leq n$, to obtain a satisfiable conjunction of constraints over the variables present in the body of rule r.
- Next, variables not present in the head of the rule are eliminated using variable (quantifier) elimination techniques to obtain a conjunction of constraints over the variables in $\overline{X}$.

An *application* of rule r consists of making all possible derivations that can be made using rule r and the set of facts known at the end of the previous iteration.

Newly computed p facts must be compared against previously computed p facts to check whether these are indeed "new" facts. This involves subsumption checks rather than equality checks (which is all that is required if each fact is a tuple of constants, as in traditional database query languages).

Note that bottom-up evaluation uses the representation of the constraint facts *directly*, instead of working with the potentially infinite set of ground facts represented by the constraint facts. The equivalence of the constraint facts computed in the bottom-up evaluation of a program P and the meaning of P in terms of its least model is in terms of the ground facts represented by the constraint facts.

THEOREM 2.1 [15]

Consider a program P and database D in a CQL with arithmetic constraints, and let $\mathcal{F}$ be the set of constraint facts computed in the bottom-up evaluation of $\langle P, D\rangle$. Let $\mathcal{M}$ be the meaning of $\langle P, D\rangle$, in terms of its least model. Then,

- (soundness) each ground instance f of a constraint fact $F \in \mathcal{F}$ is in $\mathcal{M}$, and
- (completeness) each fact f in $\mathcal{M}$ is a ground instance of a constraint fact $F \in \mathcal{F}$.

In this paper, we consider only constraint query languages with linear arithmetic constraints.

2.2. LINEAR PROGRAMMING

We assume the standard terminology of linear programming. The reader is referred to [14] for details. A few important definitions are given here.

DEFINITION 2.4: CONVEX POLYHEDRON

A set P of points in R^m is called a *convex polyhedron* if:

$$P = \{X \mid X \in R^m, AX \leq b\}$$

for some $n \times m$ matrix A and a vector b, i.e. P is the intersection of finitely many affine half-spaces.

$AX \leq b$ is said to define P. A convex polyhedron can thus be represented as a conjunction of linear arithmetic constraints; each constraint representing one of the affine half-spaces. $AX \leq b$ is said to be the *half-space* representation of a convex polyhedron.

DEFINITION 2.5: MINIMAL HALF-SPACE REPRESENTATION

In a half-space representation, $C \equiv c_1 \ \& \ldots \& \ c_m$, a constraint c_i is said to be *redundant* if $C' \equiv c_1 \ \& \ldots \& \ c_{i-1} \ \& \ c_{i+1} \ \& \ldots \& \ c_m$ represents the same convex polyhedron as C.

A half-space representation $C \equiv c_1 \ \& \ldots \& \ c_m$ of a convex polyhedron is said to be *minimal* if no c_i is redundant in C.

For instance, the constraint $X \leq 10$ is redundant in $X \leq 5 \ \& \ X \leq 10$. Thus, the half-space representation $X \leq 5 \ \& \ X \leq 10$ is not minimal, whereas the half-space representation $X \leq 5$ is minimal.

Checking whether a constraint c_i is redundant in C involves solving a linear program, and can be carried out in time polynomial in the size of the constraint set C. Further, each constraint in C needs to be considered exactly once for redundancy purposes, and hence obtaining a minimal half-space representation can be carried out in time polynomial in the size of the constraint set.

However, there need be no unique (even modulo multiplication by constants) minimal half-space representation of a convex polyhedron, as demonstrated by the following example.

EXAMPLE 2.1

Given a conjunction of constraints $X + Y + Z = 6 \ \& \ 2X + Y - Z = 2 \ \& \ 3X + 2Y = 8$ representing a convex polyhedron, the following two conjunctions are equivalent to it; each of them is also minimal: $X + Y + Z = 6 \ \& \ 2X + Y - Z = 2$ and $X + Y + Z = 6 \ \& \ 3X + 2Y = 8$.

If a convex polyhedron satisfies certain conditions, it does have a unique minimal half-space representation. However, this is not relevant to the results in this paper and we do not discuss this any further.

DEFINITION 2.6: CONTAINMENT OF A CONVEX POLYHEDRON IN A CONVEX POLYHEDRON

A convex polyhedron represented by $A_1X \leq b_1$ is said to be *contained in* another convex polyhedron represented by $A_2X \leq b_2$ if

$$\{X \mid X \in R^m, A_1X \leq b_1\} \subseteq \{X \mid X \in R^m, A_2X \leq b_2\}.$$

If this is true, we say that $A_1X \leq b_1 \subset A_2X \leq b_2$, or $A_2X \leq b_2 \supset A_1X \leq b_1$.

Given two convex polyhedra in half-space representations C_1 and C_2, we would like to determine whether $C_1 \subset C_2$. This involves solving a number of linear problems of satisfiability of conjunctions of linear arithmetic constraints. Procedure polyhedron_containment below is based on the result that $C_1 \subset C_2$ if and only if the following holds: for all selections of constraints $c_{2,j}$ from C_2, the conjunction of constraints C_1 & $\neg c_{2,j}$ is unsatisfiable.

```
polyhedron_containment (C1, C2)
{
    /* To check if C1 ⊂ C2. */
    let C1 be c1,1 & . . . & c1,m1.
    let C2 be c2,1 & . . . & c2,m2.
    for j = 1 to m2 do {
        if C1 & ¬c2,j is satisfiable, return (NOT_CONTAINED)
    }
    return (CONTAINED)
}
```

Since procedure polyhedron_containment has to solve only m_2 problems of satisfiability of conjunctions of linear constraints, each with $m_1 + 1$ constraints, it is a polynomial-time (in the size of the half-space representations of the two convex polyhedra) algorithm. Several improvements are possible to improve the efficiency of this algorithm. We do not discuss these further.

A finite union of convex polyhedra can be represented as a (finite) collection of the half-space representations of each of the constituent convex polyhedra. The collection $\{A_1X \le b_1, \ldots, A_kX \le b_k\}$ represents the union:

$$\{X \mid X \in R^m, A_1X \le b_1\} \cup \ldots \cup \{X \mid X \in R^m, A_kX \le b_k\}.$$

This union need not have a half-space representation since it may be non-convex.

For instance, the union of $X_1 \ge 4$ & $X_1 \le 5$ & $X_2 \ge 0$ & $X_2 \le 8$ and $X_1 \ge 0$ & $X_1 \le 7$ & $X_2 \ge 5$ & $X_2 \le 6$ is the non-convex region shown in fig. 1.

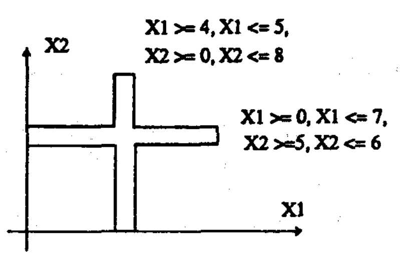

Fig. 1. Relations.

DEFINITION 2.7: CONTAINMENT IN A UNION OF CONVEX POLYHEDRA

A convex polyhedron represented by $A_0X \leq b_0$ is said to be *contained in* the union of k convex polyhedra represented by $\{A_1X \leq b_1, \ldots, A_kX \leq b_k\}$ if:

$$\{X \mid X \in R^m, A_0X \leq b_0\} \subseteq \{X \mid X \in R^m, A_1X \leq b_1\} \cup \ldots \cup \{X \mid X \in R^m, A_kX \leq b_k\}.$$

If this is true, we say that

$$A_0X \leq b_0 \subset ((A_1X \leq b_1) \vee \ldots \vee (A_kX \leq b_k)).$$

When each convex polyhedron C' in a collection $\mathscr{C}$ is contained in the union of k convex polyhedra $C_1, \ldots, C_k$, we say that

$$\mathscr{C} \subset (C_1 \vee \ldots \vee C_k).$$

DEFINITION 2.8: MINIMALITY OF REPRESENTATION

A representation $\mathscr{C} = \{C_1, \ldots, C_n\}$ of a finite union of convex polyhedra is said to be *minimal* if there is no $C_i \subset (C_1 \vee \ldots \vee C_{i-1} \vee C_{i+1} \vee \ldots \vee C_n)$.

A finite union of convex polyhedra could also have non-unique minimal representations. For instance, the union $\{X \leq 6, X \geq 5\}$ (representing the whole space) is equivalent to the union $\{X \leq 5, X \geq 4\}$ (also representing the whole space); the two representations are both minimal. Note that this non-uniqueness does not arise due to the non-uniqueness of the minimal (half-space) representation of a convex polyhedron.

In the rest of this paper, we assume that a convex polyhedron is represented in minimal half-space representation and, hence, we often identify the half-space representation of a convex polyhedron with the convex polyhedron itself. Thus, when we use "a convex polyhedron C", we mean "a convex polyhedron represented by C in half-space representation".

3. Subsumption: The problem

In this paper, we consider the problem of subsumption of a constraint fact in a CQL with linear arithmetic constraints by a relation (a finite collection of such constraint facts).

DEFINITION 3.1: SUBSUMPTION OF A CONSTRAINT FACT BY A RELATION

A constraint fact $p(\overline{X}; C)$ is said to be *subsumed* by a relation $\{p(\overline{X}; C_1), \ldots, p(\overline{X}; C_n)\}$ if each ground instance of $p(\overline{X}; C)$ is also an instance of one of the $p(\overline{X}; C_i)$, $1 \leq i \leq n$.

Checking for subsumption may make the difference between termination and non-termination of a CQL program, as the following example illustrates.

EXAMPLE 3.1 (TERMINATION VERSUS NON-TERMINATION)

Consider the CQL program P:

$r1 : e(X) :- X \leq 10, X \geq 5.$
$r2 : e(X) :- X \leq 5, X \geq 0.$
$r3 : p(X) :- e(X).$
$r4 : p(X) :- p(X1), p(X2), X = 0.5 * X1 + 0.5 * X2.$

If $r4$ is applied using $p(X; X \leq 5 \ \& \ X \geq 0)$ (computed using rules $r2$ and $r3$) in the first occurrence of p, and $p(X; X \leq 10 \ \& \ X \geq 5)$ (computed using rules $r1$ and $r3$) in the second occurrence of p, we compute the fact $p(X; X \leq 7.5 \ \& \ X \geq 2.5)$. This constraint fact can be seen to be subsumed by the collection of the two facts computed by rule $r3$. Further, it can be verified easily that each fact computed by rule $r4$ is subsumed by the collection of the facts computed by rule $r3$.

Bottom-up evaluation terminates after one iteration, is subsumption checks are performed. However, if subsumption checks are not performed, this program does not terminate.

The above example also gives some idea of the complexity of subsumption checking for programs in constraint query languages. The newly computed fact $p(X; X \leq 7.5 \ \& \ X \geq 2.5)$ using rule $r4$ is not subsumed individually by any of the facts $p(X; X \leq 5 \ \& \ X \geq 0)$ or $p(X; X \leq 10 \ \& \ X \geq 5)$, although it is subsumed by the collection of the two facts.

3.1. COMPLEXITY RESULTS

In a CQL with linear arithmetic constraints, constraint facts can be viewed geometrically as convex polyhedra, and relations can be viewed geometrically as finite (non-convex) unions of convex polyhedra. Checking whether a newly computed constraint fact is subsumed by the existing constraint facts in a relation also has a geometric interpretation. It is the problem of checking whether a convex polyhedron is contained within a finite union of convex polyhedra. The following result formalizes the relationship between containment of convex polyhedra and our original problem of subsumption of constraint facts.

THEOREM 3.1

A constraint fact $p(\overline{X}; C)$ is subsumed by a relation $\{p(\overline{X}; C_1), \ldots, p(\overline{X}; C_n)\}$ of constraint facts if and only if $C \subset (C_1 \vee \ldots \vee C_n)$.

The following results describe the complexity of the problem of containment of convex polyhedra, and hence the problem of subsumption of constraint facts.

LEMMA 3.2

Checking whether one convex polyhedron C is contained in the union of n convex polyhedra $C_1, \ldots, C_n$ is co-NP hard.

Proof

Given a Boolean formula in disjunctive normal form with at most three literals per disjunct, checking if this formula is a tautology is co-NP complete (LO8 in [5]). Call this problem 3-TAUTOLOGY. We show the co-NP hardness of checking containment by reducing 3-TAUTOLOGY to checking whether one convex polyhedron is contained in a union of convex polyhedra.

Consider a Boolean formula in disjunctive normal form with m variables $A_1, A_2, \ldots, A_m$ and n disjuncts. Associate with each variable A_i the constraint $X_i \leq 0$, and with $\overline{A_i}$ the constraint associated is $\neg(X_i \leq 0)$, i.e. $X_i > 0$. With each disjunct (which has at most three literals) we can now associate the convex polyhedron which is the intersection (in m dimensions) of the three half-spaces corresponding to each of the three literals. With the Boolean formula itself, we now associate the union of the convex polyhedra associated with each disjunct. Thus, the Boolean formula represents the union of n convex polyhedra in m dimensions.

It is easy to prove that the convex polyhedron represented as $(X_1 \leq 10 \ \& \ X_1 \geq -10 \ \& \ldots \& \ X_m \leq 10 \ \& \ X_m \geq -10)$ is contained in the union of the convex polyhedra associated with the Boolean formula if and only if the Boolean formula is a tautology. This completes the proof of the result. □

LEMMA 3.3

Checking whether one convex polyhedron C is *not* contained in the union of n convex polyhedra $C_1, \ldots, C_n$ is in NP.

Proof

A convex polyhedron C is *not* contained in the union of n convex polyhedra $C_1, \ldots, C_n$ if and only if there exists at least one convex polyhedron C' that is contained in C, and disjoint with each of the C_i, $1 \leq i \leq n$.

The oracle guesses this convex polyhedron C' (in half-space representation), and one can easily verify in polynomial time that C' is contained in C, and disjoint with each of the C_i, $1 \leq i \leq n$, by solving a polynomial number of linear programs. Further, procedure check_containment (described in section 4.3) provides a constructive proof that such a convex polyhedron can be represented (in half-space representation) in the required polynomial space. More precisely, if m_s is the space needed to represent the polyhedra $C, C_1, \ldots, C_n$, then the desired convex polyhedron C' can be represented in $O(m_s)$ space. This completes the proof of the result. □

From the above two lemmas, we obtain the result that:

THEOREM 3.4

Checking whether one convex polyhedron C is contained in the union of n convex polyhedra $C_1, \ldots, C_n$ is co-NP complete.

From the equivalence of convex polyhedra to constraint facts in a CQL with linear arithmetic constraints (theorem 3.1), we have the following corollary to theorem 3.4.

COROLLARY 3.5

Consider a program P in a CQL with linear arithmetic constraints. Checking if a constraint fact $p(\overline{X}; C)$ computed in a bottom-up evaluation of P is subsumed by the constraint facts $\{p(\overline{X}; C_1), \ldots, p(\overline{X}; C_n)\}$ in a relation is co-NP complete.

Although we described the importance of subsumption checks in constraint query languages in the context of a bottom-up evaluation, similar considerations also hold in a top-down evaluation strategy that chooses to memo the constraint facts computed, instead of recomputing them (as in CLP($\mathcal{R}$) [8], for instance). Such memoing of facts is essential for completeness with respect to the declarative semantics of CQL programs.

4. Containment of a convex polyhedron in a finite union

In section 3.1, we described the complexity of checking whether a convex polyhedron is contained in a finite union of convex polyhedra. In this section, we first describe a straightforward deterministic algorithm for this purpose, and show that it can be quite inefficient when the convex polyhedron is indeed contained in the finite union of convex polyhedra. We then present an algorithm based on the divide and conquer strategy and a linear partitioning algorithm for convex polyhedra that is often more efficient than the straightforward algorithm when the convex polyhedron is contained in the finite union.

4.1. A STRAIGHTFORWARD ALGORITHM

First, consider the simple case of a convex polyhedron represented by $C_0 \equiv c_{0,1} \,\&\, \ldots \,\&\, c_{0,m_0}$ being contained in the union of two convex polyhedra represented by $C_1 \equiv c_{1,1} \,\&\, \ldots \,\&\, c_{1,m_1}$ and $C_2 \equiv c_{2,1} \,\&\, \ldots \,\&\, c_{2,m_2}$, without being contained in either of them individually. Figure 2 illustrates this.

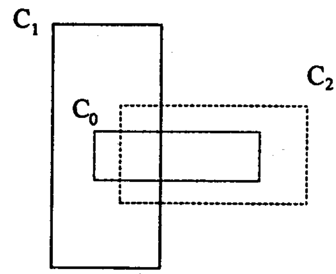

Fig. 2. A convex polyhedron contained in the union of two convex polyhedra.

This immediately suggests a mathematical way of checking this. We need to solve the following linear programs to achieve this:

$$C_0 \,\&\, \neg c_{1,i} \subset C_2, \quad 1 \leq i \leq m_1.$$

This involves m_1 calls to procedure polyhedron_containment, and determines whether the difference of the convex polyhedra $C_0 - C_1$ is contained in the convex polyhedron represented by C_2. The above set of linear programs is equivalent to checking that each of

$$C_0 \,\&\, \neg c_{1,i} \,\&\, \neg c_{2,j}, \quad 1 \leq i \leq m_1,\ 1 \leq j \leq m_2,$$

is unsatisfiable.

Consider fig. 3, where it is *not* the case that $C_0 \subset C_1 \vee C_2$. It can easily be seen how the linear programs above will determine that the polyhedron represented by C_0 is not contained in the union of the convex polyhedra represented by C_1 and C_2.

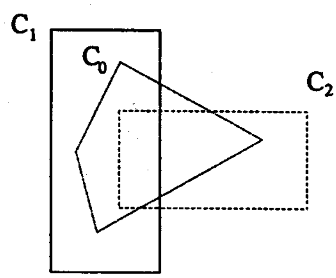

Fig. 3. A convex polyhedron not contained in the union of two convex polyhedra.

The algorithm can be extended in a straightforward fashion to determine when a convex polyhedron is contained in a union of n convex polyhedra, instead of just two convex polyhedra. Procedure straightforward_check_containment below is based on the result that C_0 is contained in the union of a finite collection of convex polyhedra, represented by $\mathscr{C}_{rel} = \{C_1, \ldots, C_n\}$ if and only if the following holds: for all selections of c_{1,j_1} from $C_1, \ldots, c_{n,j_n}$ from C_n,

$$C_0 \,\&\, \neg c_{1,j_1} \,\&\, \ldots \,\&\, \neg c_{n,j_n}$$

is unsatisfiable.

```
straightforward_check_containment (C, 𝒞_rel)
{
    let {C_1, ..., C_n} be the convex polyhedra in 𝒞_rel,
        where each C_j is of the form c_{j,1} & ... & c_{j,m_j}.
    for i_1 = 1 to m_1 do {
        :
        for i_n = 1 to m_n do {
            if (C & ¬c_{1,i_1} & ... & ¬c_{n,i_n}) is satisfiable,
                return (NOT_CONTAINED)
        }
    }
    return (CONTAINED)
}
```

PROPOSITION 4.1

Procedure straightforward_check_containment $(C, \mathscr{C}_{rel})$, where $\mathscr{C}_{rel} = \{C_1, \ldots, C_n\}$, returns CONTAINED if and only if $C \subset (C_1 \vee \ldots \vee C_n)$. Further, if m is the maximum number of constraints in C or any of the C_i in $\mathscr{C}_{rel}$, procedure straightforward_check_containment solves at most m^n problems of satisfiability of conjunctions of linear constraints, where each problem has at most $m + n$ constraints.

The main problem with procedure straightforward_check_containment is that if the convex polyhedron C *is contained in* the finite union of the convex polyhedra represented by $\mathscr{C}_{rel}$, procedure straightforward_check_containment can perform a considerable amount of unnecessary computation. This is seen in the following example.

EXAMPLE 4.1

Let C_0, C_1, C_2, C_3 and C_4 be convex polyhedra as shown in fig. 4. Each of $C_1, \ldots, C_4$ overlap with C_0, and each of the constraints in $C_1, \ldots, C_4$ is considered while checking for containment of C_0. Note that C_0 is contained within the union of just C_1 and C_2, for example. Procedure straightforward_check_containment, however, does not take advantage of such a possibility. We next present an algorithm, procedure check_containment, that does take advantage of such possibilities.

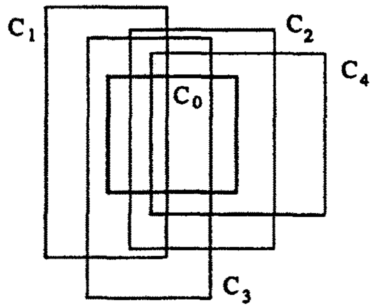

Fig. 4. Checking containment.

4.2. A LINEAR PARTITIONING ALGORITHM

We present an algorithm, linear_partition, that takes two convex polyhedra in half-space representations C_1 and C_2, with m_1 constraints and m_2 constraints, respectively, such that:

- C_1 & C_2 is satisfiable, i.e. the two convex polyhedra are not disjoint, and
- $C_2 \not\subset C_1$, i.e. the convex polyhedron represented by C_1 does not contain the convex polyhedron represented by C_2.

The algorithm then partitions the convex polyhedron represented by C_2 into $m_1 + 1$ convex polyhedra $C_{2,1}, \ldots, C_{2,m_1+1}$ such that:

- $C_{2,1} \subset C_1$, i.e. one of the convex polyhedra is contained in the convex polyhedron C_1, and
- $C_{2,i}$ & C_1 is unsatisfiable for $2 \le i \le m_1 + 1$, i.e. the convex polyhedra represented by $C_{2,2}, \ldots, C_{2,m_1+1}$ are each disjoint with the convex polyhedron represented by C_1.

This algorithm for partitioning a convex polyhedron is interesting in its own right. For example, it is used as part of a technique for optimizing queries on CQL programs in [15].

```
linear_partition (C_1, C_2)
{
    /* partition C_2 using C_1. */
    let C_1 = c_{1,1} & . . . & c_{1,m_1}
    let C_2 = c_{2,1} & . . . & c_{2,m_2}
    let C_{2,1} = C_2 & C_1   /* C_{2,1} is contained in C_1 */
    𝒞_res = ∅
    for i = 2 to m_1 + 1 do {
```

```
        let $C_{2,i} = C_2 \mathbin{\&} c_{1,1} \mathbin{\&} \ldots \mathbin{\&} c_{1,m_1+1-i} \mathbin{\&} \neg c_{1,m_1+2-i}$
        if $C_{2,i}$ is satisfiable, $\mathscr{C}_{res} = \mathscr{C}_{res} \cup \{C_{2,i}\}$
    }
    return $\mathscr{C}_{res}$
}
```

THEOREM 4.2

Consider two convex polyhedra C_1 and C_2, with m_1 and m_2 constraints, respectively, such that:

- $C_1 \mathbin{\&} C_2$ is satisfiable, and
- $C_2 \not\subset C_1$.

Then, procedure linear_partition (C_1, C_2) partitions the convex polyhedron represented by C_2 into at most $m_1 + 1$ convex polyhedra $C_{2,1}, \ldots, C_{2,m_1+1}$ such that:

(1) the convex polyhedron represented by $C_{2,1}$ is contained in the convex polyhedron represented by C_1,

(2) each convex polyhedron represented by $C_{2,i}$, $2 \leq i \leq m_1 + 1$, is disjoint with the convex polyhedron represented by C_1, and

(3) each convex polyhedron represented by $C_{2,i}$, $1 \leq i \leq m_1 + 1$, has at most $m_1 + m_2$ constraints.

Proof

We prove the theorem by proving the following claims:

Claim 1: Each $C_{2,i}$, $1 \leq i \leq m_1 + 1$, is a convex polyhedron.

Proof of claim 1: Note that the negation of a linear arithmetic constraint is also a linear arithmetic constraint. Hence, each $C_{2,i}$, $1 \leq i \leq m_1 + 1$, is a finite conjunction of linear arithmetic constraints, and this completes the proof of claim 1.

Claim 2: $C_{2,1} \subset C_1$.

Proof of claim 2: $C_{2,1} \equiv (C_2 \mathbin{\&} C_1) \subset C_1$. This completes the proof of claim 2.

Claim 3: $C_{2,i} \mathbin{\&} C_1$ is unsatisfiable for $2 \leq i \leq m_1 + 1$.

Proof of claim 3: This follows from the fact that each $C_{2,i}$, $2 \leq i \leq m_1 + 1$, has in its conjunction $\neg c_{1,j}$ as one of its constraints, and $c_{1,j} \mathbin{\&} \neg c_{1,j}$ is unsatisfiable.

Claim 4: The convex polyhedra $C_{2,i}$, $1 \leq i \leq m_1 + 1$, partition C_2.

Proof of claim 4: First, it is easy to see that

$$C_2 \equiv ((C_2 \ \& \ c_{1,1} \ \& \ldots \& \ c_{1,m_1}) \vee (C_2 \ \& \ c_{1,1} \ \& \ldots \& \ c_{1,m_1-1} \ \& \neg c_{1,m_1}) \vee \ldots \vee (C_2 \ \& \neg c_{1,1})).$$

Next, note that $C_{2,i}$ & $C_{2,j}$, $i \neq j$, is unsatisfiable since one of them has in its conjunction $c_{1,k}$ as one of its constraints, and the other has $\neg c_{1,k}$ by construction. This completes the proof of claim 4.

Claim 5: Each convex polyhedron $C_{2,i}$, $1 \le i \le m_1 + 1$, has at most $m_1 + m_2$ constraints.

Proof of claim 5: Each convex polyhedron $C_{2,i}$, $1 \le i \le m_1 + 1$, is a conjunction of constraints and has all the m_2 constraints of C_2. Further, for each $1 \le j \le m_1$, $C_{2,i}$ either has $c_{1,j}$, or $\neg c_{1,j}$, or neither of the two. No other constraints are present in the conjunction. This completes the proof of claim 5.

Claims 1–5 complete the proof of the theorem. □

Figure 5 depicts a convex polyhedron represented by B_2 being partitioned by B_1 into three convex polyhedra, represented by A_1, A_2 and A_3. The convex polyhedron represented by A_1 is contained in B_1, and each of the convex polyhedra represented by A_2 and A_3 are disjoint with B_1.

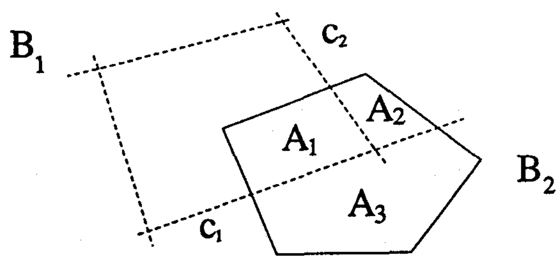

Fig. 5. Partitioning a convex polyhedron into several convex polyhedra.

Using procedure linear_partition need not result in a unique partitioning of C_2. The set of convex polyhedra actually created depends on which of the half-spaces in C_1 is treated as $c_{1,1}$, which as $c_{1,2}$ and so on. However, any partition created by procedure linear_partition satisfies theorem 4.2.

4.3. A DIVIDE AND CONQUER ALGORITHM FOR CONTAINMENT

Procedure linear_partition can be used as the basis of an algorithm to check whether a convex polyhedron represented by C is contained in the union of a collection of convex polyhedra, represented by $\mathscr{C}_{rel} = \{C_1, \ldots, C_n\}$. Procedure check_containment, which performs this check, is described below.

```
check_containment (C, 𝒞_rel)
{
  let {C_1, . . . , C_n} be the convex polyhedra in 𝒞_rel.
  𝒞_imp = {C}
  for i = 1 to n do {
    for each element C' of 𝒞_imp
      if (polyhedron_containment(C', C_i) = CONTAINED), remove C' from 𝒞_imp
    /* we have removed all convex polyhedra that are already contained */
    if (𝒞_imp = ∅) return (CONTAINED)
      /* indicates that we have successfully checked containment of C in 𝒞_rel */
    else {
      for each element C' of 𝒞_imp that is not disjoint with C_i {
        𝒞_imp = 𝒞_imp − {C'}
        𝒞_imp = 𝒞_imp ∪ linear_partition (C_i, C')
      }
    }
  }
  return (NOT_CONTAINED)
    /* when at least one of the (sub) convex polyhedra is not contained in any of
      the C_i */
}
```

An analysis of procedure check_containment shows that the cost of checking whether the convex polyhedron represented by C is contained in the union of convex polyhedra represented by $\mathcal{C}_{rel}$ depends on the maximum number of partitions of C created by the various elements in $\mathcal{C}_{rel}$, and the cost of checking whether a convex polyhedron is contained in another convex polyhedron. Containment of one convex polyhedron by another can be done in polynomial time, using procedure polyhedron_containment, for instance. All we need to know now is the maximum number of partitions that can be created from the convex polyhedron represented by C.

THEOREM 4.3

Procedure check_containment $(C, \mathcal{C}_{rel})$, where $\mathcal{C}_{rel} = \{C_1, \ldots, C_n\}$, returns CONTAINED if and only if $C \subset (C_1 \vee \ldots \vee C_n)$. If m is the maximum number of constraints in C or any other of the C_i in $\mathcal{C}_{rel}$, procedure check_containment solves $O(m^{n+1})$ problems of satisfiability of linear constraints to achieve this. Further, the size of each problem is bounded by $m + m * n$.

Proof

We prove the theorem by proving the following claims.

Claim 1: In iteration i of procedure check_containment,

$$C \subset (C_1 \vee \ldots \vee C_n) \text{ iff } \mathscr{C}_{imp} \subset (C_i \vee \ldots \vee C_n).$$

Proof of claim 1: We prove it by induction on i. Let $\mathscr{C}^i_{imp}$ denote $\mathscr{C}_{imp}$ at the beginning of iteration i. The base case is trivial. Consider the induction step. Each convex polyhedron $C' \in \mathscr{C}^i_{imp}$ such that $C' \subset C_i$ is first removed in iteration i. Next, we consider only polyhedra C' in $\mathscr{C}_{imp}$ that intersect with C_i and we partition such C' using C_i. These polyhedra C' satisfy the preconditions of theorem 4.2, and hence the partitions of C' satisfy theorem 4.2. Since each of the polyhedra added to $\mathscr{C}^i_{imp}$ are disjoint with C_i, we have

$$\mathscr{C}^{i+1}_{imp} \subset (C_{i+1} \vee \ldots \vee C_n) \text{ iff } \mathscr{C}^i_{imp} \subset (C_i \vee \ldots \vee C_n).$$

From the induction hypothesis, it follows that

$$\mathscr{C}^{i+1}_{imp} \subset (C_{i+1} \vee \ldots \vee C_n) \text{ iff } C \subset (C_1 \vee \ldots \vee C_n).$$

This completes the induction step and the proof of claim 1.

Claim 2: Procedure check_containment $(C, \mathscr{C}_{rel})$, where $\mathscr{C}_{rel} = \{C_1, \ldots, C_n\}$, returns CONTAINED if and only if $C \subset (C_1 \vee \ldots \vee C_n)$.

Proof of claim 2: Procedure check_containment $(C, \mathscr{C}_{rel})$ returns CONTAINED if and only if $\mathscr{C}_{imp}$ is empty at the end of some iteration i, $1 \leq i \leq n$. The proof of claim 2 now follows from claim 1.

Claim 3: If m is the maximum number of constraints in C or any of the C_i in $\mathscr{C}_{rel}$, $\mathscr{C}_{imp}$ has at most m^i polyhedra at the end of iteration i. Procedure check_containment solves at most $O(m^i)$ problems of satisfiability of conjunctions of linear constraints in the ith iteration. Further, the size of each of these problems is bounded by $m + i * m$.

Proof of claim 3: We prove this by induction on i. Let $\mathscr{C}^i_{imp}$ denote $\mathscr{C}_{imp}$ at the beginning of iteration i. The base case is trivial. Consider now the induction step and the ith iteration. From the induction hypothesis, there are at most m^{i-1} convex polyhedra in $\mathscr{C}^i_{imp}$, each with at most $m + (i-1) * m$ constraints. Since C_i has a maximum of m constraints, checking which of the convex polyhedra C' in $\mathscr{C}^i_{imp}$ are contained in C_i involves solving at most m^i problems of satisfiability. Further, each C' can be partitioned by C_i into at most $m + 1$ convex polyhedra, of which at most m are added to $\mathscr{C}_{imp}$. Since there were at most m^{i-1} convex polyhedra in $\mathscr{C}^i_{imp}$, there are at most m^i convex polyhedra in $\mathscr{C}^{i+1}_{imp}$. Since each convex polyhedron in $\mathscr{C}^i_{imp}$ had at most $m + (i-1) * m$ constraints, each convex polyhedron in $\mathscr{C}^{i+1}_{imp}$ has at most $m + i * m$ constraints. This completes the induction step and the proof of claim 3.

From claim 3, it follows that if m is the maximum number of constraints in any of the C_i in $\mathscr{C}_{rel}$, procedure check_containment solves at most $O(m^{n+1})$ linear programs to check for containment. This completes the proof of the second part of the theorem. The proof of the third part of the theorem follows from claim 3, and the fact that there are at most n iterations. □

Now, the worst-case time bound of procedure check_containment can be seen to be worse than the worst-case of procedure straightforward_check_containment. However, there are several situations in which procedure check_containment is better than procedure straightforward_check_containment.

Consider again example 4.1. In this example, procedure check_containment $(C_0, \mathscr{C}_{rel})$, where $\mathscr{C}_{rel} = \{C_1, C_2, C_3, C_4)$, needs to solve fewer problems of satisfiability of conjunctions of linear arithmetic constraints than procedure straightforward_check_containment. However, if C_0 was not contained in the $\mathscr{C}_{rel}$, procedure straightforward_check_containment would infer this by solving fewer problems of satisfiability of conjunctions of linear arithmetic constraints.

In general, procedure check_containment can be expected to perform better than procedure straightforward_check_containment if the convex polyhedron C is contained in the union of the convex polyhedra represented by $\mathscr{C}_{rel}$. On the other hand, if the convex polyhedron is not contained in the union of the convex polyhedra represented by $\mathscr{C}_{rel}$, procedure straightforward_check_containment can be expected to perform better. Which of these two procedures should be used depends on the newly generated constraint fact, and is outside the scope of this paper. As a heuristic, one might use a hybrid scheme where the two procedures are evaluated in an interleaved fashion until one of them returns an answer.

5. Indexing constraint facts

Recall the iterative bottom-up evaluation procedure to compute the fixpoint of a CQL program. After a rule is applied, one has to check whether each newly computed constraint fact $p(\overline{X}; C)$ is subsumed by the collection of known constraint facts $\{p(\overline{X}; C_1), \ldots, p(\overline{X}; C_n)\}$ in the p relation. Checking for subsumption can be performed either by using procedure straightforward_check_containment or by using procedure check_containment to check if the convex polyhedron represented by C is contained in the union of the convex polyhedra represented by $C_1, \ldots, C_n$.

Only those convex polyhedra C_i that intersect with (equivalently, are not disjoint with) C need be used to check for containment. This involves solving several problems of satisfiability of conjunctions of linear constraints – and, although this can be performed in polynomial time, it could dominate the overall cost of containment if the number of constraint facts in the p relation is very large (as is the case in database applications), and only a few of the convex polyhedra intersect with C. Consequently, we need an indexing mechanism to efficiently eliminate a large number of convex polyhedra that do not intersect with C.

R-trees (Guttman [6]) and R^+-trees (Sellis et al. [16]) have been proposed as dynamic index structures to efficiently index spatial data. The index record entries for these index structures are m-dimensional rectangles of the form

$$I = (I_1, \ldots, I_m),$$

where each I_i is a closed bounded interval $[a_i, b_i]$ describing the extent of the object along dimension i. Alternatively, one or both of a_i and/or b_i may be infinity, indicating that the object extends indefinitely. These index structures can be used to efficiently index constraint facts in a constraint query language with linear arithmetic constraints (equivalently, convex polyhedra) by associating a bounding box with each constraint fact.

While the indexing strategies described are themselves not novel (since we adapt them from indexing strategies for spatial databases), the idea of using these strategies for indexing constraint facts in the bottom-up evaluation of a CQL program is a novel contribution.

5.1. USING MINIMUM BOUNDING BOXES

DEFINITION 5.1: MINIMAL BOUNDING BOX

Given a convex polyhedron C, a bounding box BB for C is said to be a *minimal bounding box* if there does not exist a bounding box BB' for C such that (1) BB is also a bounding box for BB', and (2) BB' is not a bounding box for BB. □

The existence of a unique *minimum* bounding box for a convex polyhedron is guaranteed by the following proposition.

PROPOSITION 5.1

Given a convex polyhedron C, if BB_1 and BB_2 are bounding boxes for the convex polyhedron C, then the intersection of bounding boxes BB_1 and BB_2 is also a bounding box for the convex polyhedron C.

Minimum bounding boxes have several nice properties[1]:

Intersection: If two convex polyhedra C_1 and C_2 intersect, so do their minimum bounding boxes.

Containment: If a convex polyhedron C_1 is contained in convex polyhedron C_2, the minimum bounding box BB_{C_1} for C_1 is also contained in the minimum bounding box BB_{C_2} for C_2.

[1] Note that property *intersection* is also satisfied by non-minimum bounding boxes for convex polyhedra, although property *containment* may not be.

If the half-space representation is chosen for convex polyhedra, obtaining a minimum bounding box BB_C involves solving $2 * m$ linear programs, where m is the number of dimensions of the convex polyhedron. Minimum bounding boxes can be used to efficiently eliminate a large number of convex polyhedra in the collection $\mathscr{C}$ that do not intersect with a given convex polyhedron C. This is done by associating the minimum bounding box with each convex polyhedron, and maintaining the bounding boxes as an R⁺-tree, for example. Procedure index_constraint_facts below describes this algorithmically.

```
index_constraint_facts (C, 𝒞)
{
  let 𝒞 be {C_1, ..., C_n}.
    /* we need those C_i which intersect with C. */
  let BB_𝒞 be {BB_{C_1}, ..., BB_{C_n}} be the minimum bounding boxes
    for C_1, ..., C_n maintained as an R+-tree.
    /* these are assumed to be known */
  compute BB_C, the minimum bounding box for convex polyhedron C.
  use the R+-tree data structure to efficiently retrieve those BB_{C_i} that intersect with
  BB_C.
  return those C_i whose minimum bounding boxes BB_{C_i} intersect with BB_C.
}
```

THEOREM 5.2

Consider a convex polyhedron C, and a finite collection of convex polyhedra $\mathscr{C}$. Let $\mathscr{C}_1$ be the result of applying procedure index_constraint_facts $(C, \mathscr{C})$. Then,

$$C \subset \mathscr{C} \text{ iff } C \subset \mathscr{C}_1.$$

Proof

The proof follows from property *intersection* of bounding boxes, which states that if the minimum bounding boxes of two convex polyhedra do not intersect, then neither do the convex polyhedra. □

5.2. USING NON-MINIMUM BOUNDING BOXES

Recall that our motivation for indexing convex polyhedra was to eliminate constraint facts while checking for subsumption of a newly generated constraint fact in the bottom-up evaluation of a CQL program. However, in computing the minimum bounding box BB_C for the convex polyhedron C corresponding to the newly computed p fact, procedure index_constraint_facts does not make use of the bounding boxes for the convex polyhedra corresponding to the p_i facts used to compute this p fact (definition 2.3 describes rule applications). However, these are available (by assumption) and one could use the bounding boxes corresponding to the body facts to efficiently compute a bounding box corresponding to the newly computed head constraint fact based on the following two results:

PROPOSITION 5.3

Consider two convex polyhedra C_1 and C_2 in m dimensions. Let $C_3 = C_1$ & C_2 represent the intersection of these two convex polyhedra. Let BB_{C_1} and BB_{C_2} be bounding boxes for C_1 and C_2, respectively, and let BB_{C_3} be the intersection of the two bounding boxes.

Then, BB_{C_3} is a bounding box for C_3. Further, BB_{C_3} can be computed in time $O(m)$.

Note that bounding box BB_{C_3} need not be the minimum bounding box for C_3 even if BB_{C_1} and BB_{C_2} are the minimum boxes for C_1 and C_2, respectively. This can be seen from fig. 6. However, non-minimum bounding boxes still satisfy property *intersection*, i.e. if two convex polyhedra intersect, so do their bounding boxes.

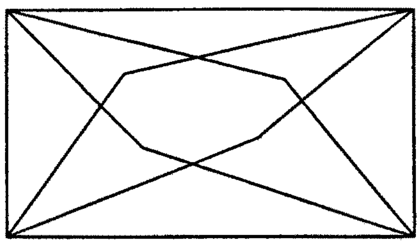

Fig. 6. Bounding boxes for convex polyhedra.

PROPOSITION 5.4

Consider a convex polyhedron C_1 in m dimensions, and let C_2 be its projection onto $k < m$ dimensions. Let BB_{C_1} be the minimum bounding box of C_1, and BB_{C_2} be the projection of BB_{C_1} onto to the corresponding k dimensions.

Then, BB_{C_2} is the *minimum* bounding box for C_2. Further, BB_{C_2} can be computed in $O(m)$.

The results of propositions 5.3 and 5.4 can be used to efficiently compute a bounding box, though not the minimum possible, for a newly computed constraint fact, using bounding boxes for the facts used in the body to compute the head constraint fact. Procedure compute_bounding_box below describes this algorithmically.

```
compute_bounding_box (r, p_1(X̄_1; C_1), ..., p_n(X̄_n; C_n))
{
  let r be the rule:
    r : p(X̄) :- C, p_1(X̄_1), ..., p_n(X̄_n).
```

```
    let p_i(X_i; C_i) be the constraint fact used in atom p_i(X_i).
    let BB_C1, ..., BB_Cn be bounding boxes for C_1, ..., C_n.
      /* these are assumed to have been previously computed. */
    let m be the number of variables in rule r.
    extend each BB_Ci to all m dimensions.
    compute BB = ∩_{i=1}^{n} (BB_Ci).
    let k be the number of variables in the head.
    compute BB_C = Π_k(BB).
      /* BB_C is a bounding box for the newly computed p fact. */
    return BB_C.
}
```

Although the bounding box computed by procedure compute_bounding_box is not the minimum bounding box, it still satisfies property *intersection*. Consequently, if procedure compute_bounding_box is used to compute the bounding box in procedure index_constraint_facts, the resulting algorithm will still satisfy theorem 5.2. This modified algorithm can be used to efficiently eliminate some convex polyhedra in $\mathscr{C}$ that do not intersect with C. However, since procedure compute_bounding_box does not compute minimum bounding boxes, we may not be able to eliminate as many convex polyhedra as were eliminated by using procedure index_constraint_facts directly. Thus, we have a trade-off between efficiently computing a bounding box for C versus efficiently eliminating a large number of convex polyhedra in $\mathscr{C}$, and each strategy may be more efficient in certain situations. Which method to adopt depends on the constraint facts in the p relation in question, and is outside the scope of this paper.

6. Polynomial time sufficient conditions

Checking whether a convex polyhedron is contained in a finite union of convex polyhedra is extremely expensive, in general. Since the problem is co-NP complete, there is no polynomial time deterministic solution for the problem (unless P = NP), in general. In this section, we describe conditions under which containment or non-containment can be checked in polynomial time.

6.1. CHECKING CONTAINMENT

If all the constraints in a convex polyhedron are equality constraints, the convex polyhedron is a finite conjunction of hyperplanes, and is affine. For such convex polyhedra, containment can be checked in polynomial time. This leads to the following interesting restriction on CQL programs for which subsumption can be checked in polynomial time.

PROPOSITION 6.1

Consider a constraint query language where only linear arithmetic equality constraints are permitted. In such a CQL, a constraint fact $p(\overline{X}; C)$ is subsumed by a finite collection of constraint facts $\{p(\overline{X}; C_1), \ldots, p(\overline{X}; C_n)\}$ if and only if it is subsumed by $p(\overline{X}; C_i)$ for some $1 \leq i \leq n$.

As a consequence of the above result, subsumption of $p(\overline{X}; C)$ needs to be checked against only one constraint fact in the p relation at a time. Consequently, subsumption can be checked in polynomial time.

This result is important from the perspective of evaluating CQL programs. Given a program with only linear arithmetic equality constraints – and this can be checked at compile time – one need check for subsumption of a newly generated constraint fact against only one fact at a time, considerably improving the efficiency of bottom-up evaluation of CQL programs.

6.2. CHECKING NON-CONTAINMENT

We describe two simple polynomial-time sufficient conditions that allow us to check that a convex polyhedron is not contained in a finite union of convex polyhedra.

The first one is based on checking that the given convex polyhedron is not contained in a sufficiently large convex polyhedron.

Using procedure index_constraint_facts, we can efficiently eliminate a large number of convex polyhedra that do not intersect with C, the polyhedron that is being checked for containment in the union of convex polyhedra represented by $\mathscr{C}$. Let $\mathscr{C}_1 = \{C_1, \ldots, C_k\}$ be the result of applying procedure index_constraint_facts $(C, \mathscr{C})$. Let BB_{C_j}, $1 \leq j \leq k$, be the corresponding minimum bounding boxes, each an m-dimensional rectangle.

Let BB be the minimum m-dimensional rectangle that contains each of BB_{C_j}. Such an m-dimensional rectangle can be obtained in time $2 * m * k$ be examining each of the k bounding boxes, and finding the minimum and maximum value along each of the m dimensions. Bounding box BB can be used to check for non-containment in polynomial time, as the following result indicates.

PROPOSITION 6.2

Consider a convex polyhedron C, and a finite collection of convex polyhedra $\mathscr{C}$. Let $\mathscr{C}_1$ be the convex polyhedra obtained using procedure index_constraint_facts $(C, \mathscr{C})$. Also, let BB be a bounding box that contains each of the bounding boxes of the convex polyhedra in $\mathscr{C}_1$. Then,

$$\text{if } C \not\subset BB \text{ then } C \not\subset \mathscr{C}.$$

Further, this check for non-subsumption can be performed in polynomial time.

The result follows from the properties of bounding boxes.

The second polynomial-time sufficient condition for non-containment is based on obtaining points on the surface of the given convex polyhedron C and checking that at least one of these points is not contained in each C' in $\mathcal{C}_1$, the convex polyhedra obtained using procedure index_constraint_facts. As a heuristic, we suggest obtaining such points by maximizing or minimizing (in the feasible region given by C) each of the objective functions obtained by considering the constraints of each convex polyhedron in $\mathcal{C}_1$.

PROPOSITION 6.3

Consider a convex polyhedron C, and a finite collection of convex polyhedra $\mathcal{C}$. Let $\mathcal{C}_1 = \{C_1, \ldots, C_k\}$ be the convex polyhedra obtained using procedure index_constraint_facts $(C, \mathcal{C})$. Let $\mathcal{P} = P_1, \ldots, P_l$ be a collection of points on the surface of C obtained by maximizing or minimizing each of the objective functions obtained by considering the constraints of each of the C_i. Then,

$$\text{if } \exists j, \text{ such that } (P_j \not\subset C_1) \,\&\, \ldots \,\&\, (P_j \not\subset C_k) \text{ then } C \not\subset \mathcal{C}.$$

Further, this check for non-subsumption can be performed in time polynomial in the size of the representations of $C, C_1, \ldots, C_k$.

The proof of the time complexity follows from the fact that there are only a polynomial number of objective functions we consider for this heuristic.

Again, this is just a sufficient condition since each surface of a convex polyhedron represented by C may be contained in the union of convex polyhedra represented by $\mathcal{C}$, but C may still not be contained in $\mathcal{C}$. An example is given in fig. 7.

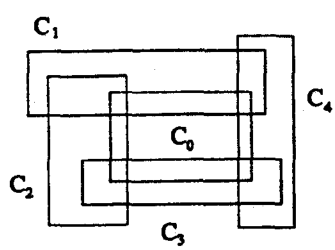

Fig. 7. Surfaces of a convex polyhedron contained.

7. Incrementally checking for containment

Recall the iterative bottom-up evaluation procedure for computing the fixpoint of a CQL program. In each iteration, the procedure first applied the program rules to compute constraint facts as follows: Consider the program rule

$$r : p(\overline{X}) :- C, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).$$

Given a p_i fact for each $p_i(\overline{X_i})$, $1 \leq i \leq n$, we have a conjunction of constraints over the variables present in the body of r. To obtain a p fact using these body facts and rule r, variables not present in the head of rule r need to be projected out. Then the bottom-up evaluation procedure checked whether each newly computed constraint fact was subsumed or was a "new" constraint fact. Computing constraint facts is quite expensive because of the projection operation [11] used in computing the head fact. If the newly computed constraint fact *is subsumed*, the cost of computing this constraint fact is "wasted".

We now describe an algorithm that interleaves the computation of constraint facts with checking for subsumption in the bottom-up evaluation of a CQL program. This algorithm tries to minimize the wasted effort by early detection of when a newly computed constraint fact is subsumed. It is based on a recent algorithm by Lassez and Lassez [10] for quantifier (variable) elimination for systems of linear constraints. The algorithm in [10] computes successive approximations using linear programming techniques and an on-line convex hull construction algorithm. Further, the algorithm can provide an upper bound or lower bound approximation or both.

We use the algorithm in [10] as the basis of an incremental algorithm to check for subsumption. Given this algorithm, we can perform the projection operation used in computing a constraint fact in an incremental fashion, to obtain better and better upper bound approximations to the actual p fact that follows from the given body facts and the rule r. This can be used to advantage in incrementally checking for containment. Procedure incremental_containment below performs this incremental check.

incremental_containment $(C(\overline{Y}), \overline{X}, \mathscr{C}_{rel})$
{
  $C(\overline{Y})$ is the convex polyhedron that needs to be projected onto $\overline{X}$.
  the resultant polyhedron C_p needs to be checked for containment in $\mathscr{C}_{rel}$.
  let C_p^1 be a first upper-bound approximation to C_p obtained using the algorithm of [10].
  partition C_p^1 using $\mathscr{C}_{rel}$ (and linear_partition) into two sets of convex polyhedra,
    $\mathscr{C}_{subs}$: where each polyhedron in $\mathscr{C}_{subs}$ is contained in $\mathscr{C}_{rel}$, and
    $\mathscr{C}_{disj}$: where each polyhedron in $\mathscr{C}_{disj}$ is disjoint from $\mathscr{C}_{rel}$.
  repeat
    if $\mathscr{C}_{disj} = \emptyset$ return (CONTAINED)
      let C_p^j be the next upper-bound approximation obtained. /* $C_p^j \subset C_p^{j-1}$. */
      let $\mathscr{C}_{disj} = \{C_1, \ldots, C_k\}$.
      for $i = 1$ to k do
        if C_i & C_p^j is unsatisfiable, discard C_i from $\mathscr{C}_{disj}$.
        else replace C_i in $\mathscr{C}_{disj}$ by C_i & C_p^j.

```
  until C_p^j = C_p.
  if 𝒞_disj = ∅ return (CONTAINED)
  else return (NOT_CONTAINED)
}
```

THEOREM 7.1

If the algorithm in [10] terminates, procedure incremental_containment $(C(\overline{Y}), \overline{X}, \mathscr{C}_{rel})$ returns CONTAINED if and only if the convex polyhedron that is the projection of $C(\overline{Y})$ on the variables in $\overline{X}$ is contained in the union of the convex polyhedra represented by $\mathscr{C}_{rel}$.

Proof

Let C_p be the projection of $C(\overline{Y})$ on the variables in $\overline{X}$, and let $\mathscr{C}_{rel}$ be the convex polyhedra corresponding to the previously computed p facts. Let C_p^i denote the upper bound approximation to C_p at the ith stage of the algorithm of [10]. The upper-bound approximations satisfy the following properties:

(1) $\forall i,\ C_p^{i+1} \subset C_p^i$, and
(2) $\forall i,\ C_p \subset C_p^i$.

We prove the theorem by proving the following claims:

Claim 1: Consider a partition of C_p^i using $\mathscr{C}_{rel}$. Let $\mathscr{C}_{subs}^i$ be the convex polyhedra in the partition contained in $\mathscr{C}_{rel}$, and $\mathscr{C}_{disj}^i$ be the convex polyhedra in the partition disjoint with the polyhedra in $\mathscr{C}_{rel}$. Then,

(1) each convex polyhedron in $\mathscr{C}_{disj}^i$ is contained in the union of the convex polyhedra in $\mathscr{C}_{disj}^j$, $j < i$, and
(2) each convex polyhedron in $\mathscr{C}_{subs}^i$ is contained in the union of the convex polyhedra in $\mathscr{C}_{subs}^j$, $j < i$.

Proof of claim 1: The proof of each part of claim 1 follows from the fact that each C_p^i is contained in each C_p^j, $j < i$, i.e. the successive approximations to C_p are better and better upper-bound approximations.

Claim 2: Procedure incremental_containment returns CONTAINED in the ith iteration if and only if $C_p^i \subset \mathscr{C}_{rel}$.

Proof of claim 2: We prove this by induction on i. The base case follows trivially from theorem 4.3. Consider the induction step and the ith iteration. Assume that procedure incremental_containment has not returned CONTAINED prior to iteration i. If procedure incremental_containment returns CONTAINED, it is because $\mathscr{C}^i_{disj} = \emptyset$. Since $\mathscr{C}^i_{subs} \subset \mathscr{C}^{i-1}_{subs}$ (from claim 1), it follows that $C_p^i \subset \mathscr{C}_{rel}$.

If procedure incremental_containment does not return CONTAINED, it is because $\mathscr{C}^i_{disj} \neq \emptyset$. Thus, there must be some convex polyhedron C' in $\mathscr{C}^i_{disj}$ such that $C' \,\&\, C^i_p$ is satisfiable. Since each C' in $\mathscr{C}^i_{disj}$ is disjoint from each of the convex polyhedra in $\mathscr{C}_{rel}$, it follows that $C^i_p \not\subset \mathscr{C}_{rel}$. This completes the proof of claim 2.

Claim 3: If the algorithm in [10] terminates, procedure incremental_containment returns CONTAINED if and only if $C_p \subset \mathscr{C}_{rel}$.

Proof of claim 3: If procedure incremental_containment returns CONTAINED, it returns it in some iteration i. From claim 2, we know that it must be the case that $C^i_p \subset \mathscr{C}_{rel}$. Since C^i_p is an upper-bound approximation to C_p, $C_p \subset C^i_p$. This completes the "only if" part of the proof. Since the algorithm in [10] for quantifier elimination terminates, from the precondition of the claim it must terminate in some iteration i. The proof of the "if" part now follows from claim 2.

This completes the proof of the theorem. □

Note that procedure incremental_containment uses linear_partition only once, on the first upper-bound approximation, C^1_p to C_p. Subsequently, we only find the intersection of each convex polyhedron in $\mathscr{C}_{disj}$ with the next (better) upper-bound approximation. If in each iteration, $\mathscr{C}_{rel}$ was used to partition C^i_p, the algorithm would repeat a lot of work, and would not be truly incremental.

Procedure incremental_containment can be used as the basis of an incremental procedure for bottom-up evaluation of CQL programs. We check whether C_p is contained in $\mathscr{C}_{rel}$ using the sequence of approximations C^i_p instead. If an upper-bound approximation to C_p is contained in the union of the existing p facts, the p fact represented by C_p will be eliminated. Hence, one does not have to continue computing further upper-bound approximations, $C^{i+1}_p, C^{i+2}_p, \ldots$ to C_p. This can improve the efficiency of evaluation, by preventing considerable wasted effort.

However, if C_p is not contained in the union of the existing C_p facts, the interleaved algorithm is more time consuming than just the serial application of variable elimination and checking containment. One can modify procedure incremental_containment to also check for non-containment using procedure straightforward_check_containment and lower-bound approximations obtained using the algorithm of [10]. Deciding when this interleaved algorithm is worthwhile depends on the constraint facts generated, and is outside the scope of this paper.

8. Conclusions and future work

We considered the problem of subsumption in constraint query languages with linear arithmetic constraints and showed the problem to be co-NP complete. We presented a deterministic algorithm based on the divide and conquer strategy to solve this problem. We identified a class of constraint query languages where subsumption can be checked in polynomial time, and suggested polynomial-time heuristics to check that a constraint fact is not subsumed by a relation. We adapted index structures for spatial data to efficiently index constraint facts in a CQL with linear arithmetic constraints. To our knowledge, this is the first indexing scheme described for this class of constraint facts. We also described an incremental scheme for bottom-up evaluation of CQL programs that interleaves approximate computation of constraint facts with an incremental check for subsumption.

There are several interesting directions of future research. One of the most interesting directions is to determine the conditions best suited for each of the containment algorithms described in this paper. Another direction is to identify larger classes of constraint query languages where subsumption is in polynomial time. Identifying efficient polynomial-time heuristics to check for non-subsumption is also practically useful. Index structures for spatial data typically support a larger class of operations than are required in the bottom-up evaluation of CQL programs. A promising direction of research is to come up with more efficient indexing strategies for constraint facts given the set of desired operations on these facts.

Acknowledgements

We would like to thank Olvi Mangasarian, Raghu Ramakrishnan and Peter Stuckey for valuable discussions. We would also like to thank the referees for providing several valuable suggestions to improve the content and presentation of this paper.

References

[1] F. Bancilhon, Naive evaluation of recursively defined relations, in: *On Knowledge Base Management Systems – Integrating Database and AI Systems*, ed. Brodie and Mylopoulos (Springer, 1985).

[2] M. Baudinet, M. Niezette and P. Wolper, On the representation of infinite temporal data and queries, in: *Proc. 10th ACM Symp. on Principles of Database Systems*, Denver, CO (1991) pp. 280–290.

[3] I. Balbin and K. Ramamohanarao, A generalization of the differential approach to recursive query evaluation, J. Logic. Progr. 4(3) (1987).

[4] J. Chomicki, Polynomial time query processing in temporal deductive databases, in: *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, TN (1990) pp. 379–391.

[5] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, 1979).

[6] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: *Proc. ACM SIGMOD Conf. on Management of Data* (1984) pp. 47–57.

[7] J. Jaffar and J.-L. Lassez, Constraint logic programming, in: *Proc. 14th ACM POPL*, Munich (1987) pp. 111–119.

[8] J. Jaffar, S. Michaylov, P. Stuckey and R. Yap, The CLP($\Re$) language and system, Technical Report, IBM, T.J. Watson Research Center (1990).

[9] P.C. Kanellakis, G.M. Kuper and P.Z. Revesz, Constraint query languages, in: *Proc. 9th ACM Symp. on Principles of Database Systems* Nashville, TN (1990) pp. 299–313.

[10] C. Lassez and J.-L. Lassez, Quantifier elimination for conjunctions on linear constraints via a convex hull algorithm, submitted.

[11] J.-L. Lassez and M.J. Maher, On Fourier's algorithm for linear arithmetic constraints, Technical Report, IBM, T.J. Watson Research Center (1988).
[12] R. Ramakrishnan, Magic templates: A spellbinding approach to logic programs, in: *Proc. Int. Conf. on Logic Programming*, Seattle, Washington (1988) pp. 140–159.
[13] P.Z. Revesz, A closed form for Datalog queries with integer order, in: *Int. Conf. on Database Theory*, France (1990) pp. 187–210.
[14] A. Schrijver, *Theory of Linear and Integer Programming*, Discr. Math. Optim. (Wiley–Interscience, 1986).
[15] D. Srivastava and R. Ramakrishnan, Pushing constraint selections, in: *Proc. 11th ACM Symp. on Principles of Database Systems*, San Diego, CA (1992).
[16] T. Sellis, N. Roussopoulos and C. Faloutsos, The R^+-tree: A dynamic index for multi-dimensional objects, in: *Proc. 13th Int. Conf. on Very Large Databases* (1987) pp. 507–518.