Constraint-Based Query Optimization for Spatial Databases

Richard Helm, Kim Marriott, Martin Odersky

I.B.M. Thomas J. Watson Research Center
P.O. Box 704, Yorktown Heights
NY 10598, USA

Abstract

We present a method for converting a system of multivariate Boolean constraints into a sequence of univariate range queries of the type supported by current spatial databases. The method relies on the transformation of a Boolean constraint system into triangular form. We extend previous results in this area by considering negative as well as positive constraints. We also present a method to approximate triangular Boolean constraints by bounding box constraints.

1 Introduction

In spatial database systems, there is a gap between the high-level query language required by applications and users, and the simpler query language supported by the underlying spatial data-structure. Typically, applications such as geographic information systems [5, 8, 10], visual language parsers [7], VLSI design rule checkers [14], require a query language in which queries and integrity constraints may be expressed over a number of variables subject to *Boolean constraints* (that is, constraints over sets). In contrast, spatial data-structures [6, 9, 12, 13] generally support only *range queries*. These are queries over a single unknown variable x of the form $x \subseteq a, b \subseteq x, x \cdot c \neq \emptyset$ where a, b, and c are given *bounding boxes*. (A bounding box is a rectangular region with sides parallel to the axes). Here, we give a query optimization technique to bridge this gap between Boolean constraints over many variables and range queries over a single variable.

The essential idea behind the optimization is to use the Boolean constraints to eliminate useless partial solution tuples as soon as possible. Consider a query involving variables $x_1, ..., x_n$ constrained by Boolean constraints C. We wish to find assignments of objects from the database to the variables which satisfy the constraints. That is, we wish to find *solution tuples* $\langle a_1, ..., a_n \rangle$ such that each a_i is in the database and $C \wedge x_1 = a_1 \wedge ... \wedge x_n = a_n$ holds. In our approach, the set of solution tuples is constructed incrementally as follows. We find the partial solution tuples $\langle a_1 \rangle$ for x_1, then the tuples $\langle a_1, a_2 \rangle$ for x_1 and x_2, and so on until the solution tuples $\langle a_1, a_2, ..., a_n \rangle$ to the query are obtained. At each step, the constraints C can be used to eliminate useless partial solution tuples in two ways. First, we need only keep those partial solutions for which there is some possible assignment to the remaining unknown variables which satisfies C. Thus we need only keep those $\langle a_1, a_2, ..., a_i \rangle$ for which $\exists x_{i+1} \ldots \exists x_n . C'$ is satisfiable, where C' is obtained by replacing each known x_j by a_j. Second, when retrieving objects a_i from the database to join to the tuple $\langle a_1, a_2, ..., a_{i-1} \rangle$, we use a range query which approximates $\exists x_{i+1} \ldots \exists x_n . C'$ to *filter* the choices for x_i. This approach requires us to compute the following "triangular solved form" for C:

$$\begin{array}{l} C_1(x_1) \\ C_2(x_1, x_2) \\ \quad\vdots \\ C_n(x_1, x_2 ..., x_n). \end{array}$$

Each C_i is a conjunction of (unquantified) Boolean

0

constraints which is equivalent to the formula $\exists x_{i+1}$. ...$\exists x_n . C$ or is some approximation to this formula. Note that this is similar to the solved form obtained using Gaussian elimination in equations over the real numbers. Furthermore, we must be able to find a range query which approximates each C_i. Effective methods to compute the triangular solved form and the approximating range queries are the main contribution of this paper.

We consider Boolean constraints as our high-level query language. We allow both positive constraints of the form $f \subseteq g$, and negative constraints of the form $f \not\subseteq g$ where f and g are Boolean formulas. These are sufficient to provide equality, dis-equality and strict containment, as

$$\begin{array}{lcl} x = y & \Leftrightarrow & x \subseteq y \wedge y \subseteq x, \\ x \subset y & \Leftrightarrow & x \subseteq y \wedge y \not\subseteq x, \\ x \neq y & \Leftrightarrow & x \cdot \overline{y} + \overline{x} \cdot y \not\subseteq \emptyset. \end{array}$$

Although systems of positive Boolean constraints have been extensively studied since Boole [2], see for example [3] or [11], the extension to negative constraints has not, to our knowledge, been addressed. This may be because in the case of two-valued Boolean algebras, negative constraints add no power since the constraint $x \not\subseteq y$ is equivalent to $x = 1 \wedge y = \emptyset$. For more general Boolean algebras, however, systems of arbitrary Boolean constraints are strictly more powerful than systems of positive constraints.

Our main technical results can be summarized as follows:

- In general, systems of Boolean constraints are not closed under existential quantification. However, we give an effective method of computing the best approximation to an existentially quantified system.
- A special – and in our setting quite natural – class of Boolean algebras, which includes the measurable subsets of R^k, is closed under existential quantification. In these algebras, the computed approximation is exactly the existentially quantified system.
- The best approximation, C_i, to each $\exists x_{i+1}, ..., \exists x_n . C$ can be expressed in solved form:

  $$s \subseteq x_i \subseteq t \wedge g_1(x_i) \neq \emptyset \wedge \ldots \wedge g_m(x_i) \neq \emptyset$$

  where each $g_i(x_i)$ is of the form $x_i \cdot p + \overline{x_i} \cdot q$, and p, q, s and t are Boolean functions over $x_1, ..., x_{i-1}$. For systems in solved form, we give a method of effectively computing the best approximating range query.

The only other approach that we are aware of to application independent multivariable spatial queries is that of Orenstein and Manola [10]. Their query language provides a spatial join, a binary overlay constraint, which can be efficiently implemented using z-order based methods. The query language we consider is more expressive because arbitrary Boolean constraints are allowed. Furthermore, our technique does not require a special purpose data structure. However, it seems possible to extend our approach to make use of z-ordering methods.

This paper is organized as follows. Section 2 illustrates our approach with an example. Section 3 gives a method for computing the triangular solved form of a Boolean constraint system. Section 4 gives a method to approximate this system by bounding box constraints. Section 5 concludes.

2 Example

Let's assume smugglers are "importing" prohibited goods into a given country C, and wish to know where to site their distribution operation. The goods must be imported at some border town T, and transported into some destination area A in C. Assume further that, while it is relatively easy to enter the country C, there are massive police patrols along the country's internal state boundaries. The transport of the prohibited goods is safe as long as no internal state boundary is crossed. Hence, the smugglers want to find a border town T, and a road R, from T to A, which does not cross a state boundary between T and A, that is, which proceeds entirely within some state B. Assuming the smugglers have access to a spatial database, they could formalize their problem as given in Figure 1.

$$\begin{array}{lcl} A & \subseteq & C \\ B & \subseteq & C \\ R & \subseteq & A+B+T \\ R\cdot A & \neq & \emptyset \\ R\cdot T & \neq & \emptyset \\ T & \not\subseteq & C. \end{array}$$

Figure 1: A system of Boolean constraints

We can re-write this system into a single equation and three disequations:

$$A\cdot\overline{C}+B\cdot\overline{C}+R\cdot\overline{A}\cdot\overline{B}\cdot\overline{T}=\emptyset,$$
$$R\cdot A\neq\emptyset, R\cdot T\neq\emptyset, \overline{C}\cdot T\neq\emptyset.$$

Assume we are given C and A, then T, R, and B must be found. We arbitrarily pick the retrieval order T, R, B. That is, we first search for the border town, then the road, and finally for the state. Then, using the methods described in the next section, we can convert the constraint system of Figure 1 into the triangular form:

$$\begin{array}{rcccll} \emptyset & \subseteq & T & \subseteq & 1, & \overline{C}\cdot T\neq\emptyset \\ \emptyset & \subseteq & R & \subseteq & C+T, & A\cdot R\neq\emptyset, \\ & & & & & R\cdot T\neq\emptyset \\ R\cdot\overline{A}\cdot\overline{T} & \subseteq & B & \subseteq & C. & \end{array}$$

Note that every variable is constrained only by variables which precede it in the retrieval order. This means that a range query can in principle be used for every retrieval step. For example, T is accessed using the range query $\lceil\overline{C}\rceil\sqcap\lceil T\rceil\neq\emptyset$ where $\lceil f\rceil$ is the minimal surrounding bounding box of f and $\sqcap$ denotes bounding box intersection. However, this would entail the precise computation of complements and intersections of arbitrary regions, a potentially expensive operation. Alternatively, using the methods of Section 4, we can approximate the triangular Boolean system by a system solely defined in terms of bounding boxes. The result of this step is:

$$\begin{array}{lllll} \emptyset & \subseteq & \lceil T\rceil & \subseteq & 1, & \\ \emptyset & \subseteq & \lceil R\rceil & \subseteq & \lceil C\rceil\sqcup\lceil T\rceil, & \lceil A\rceil\sqcap\lceil R\rceil\neq\emptyset, \\ & & & & & \lceil R\rceil\sqcap\lceil T\rceil\neq\emptyset \\ \emptyset & \subseteq & \lceil B\rceil & \subseteq & \lceil C\rceil. & \end{array}$$

Every line of this system can be implemented by a single range query which combines containment and overlap constraints on bounding boxes.

3 Systems of Boolean Constraints

Consider a system of Boolean constraints S in variables $x_1,...,x_n$. In this section we will develop an effective method of computing the following *triangular solved form* of S:

$$\begin{array}{l} C_1(x_1) \\ C_2(x_1,x_2) \\ \quad\vdots \\ C_n(x_1,x_2,...,x_n). \end{array}$$

Each C_i is the strongest system of Boolean constraints which is a necessary condition for $x_1,...,x_i$ to be a solution of S. Each C_i is of the form:

$$s(x_1,...,x_{i-1})\subseteq x_i\subseteq t(x_1,...,x_{i-1})\wedge \quad r_1\neq\emptyset\wedge...\wedge r_m\neq\emptyset \tag{1}$$

where each r_j has the form

$$x_i\cdot p(x_1,...,x_{i-1})+\overline{x_i}\cdot q(x_1,...,x_{i-1}).$$

The method to find the triangular solved form is obtained in two stages. In the first stage, we develop a method to find a system which is a maximal necessary condition for $\exists x_{i+1}\,.\,\exists x_{i+2}\ldots\exists x_n\,.\,S$. This is complicated by the fact that Boolean constraints are not closed under existential quantification. However, we give a way to find the best approximating unquantified system. Furthermore, for a class of reasonable Boolean algebras, namely the *atomless* algebras, this approximation is equivalent to the existentially quantified system. One example of an atomless algebra which is important in a spatial database context are the measurable sets in $\Re^k$. In the second stage we show how to use this approximation to transform a system of Boolean constraints into solved form.

First, some preliminary definitions: An *atom* is a variable or a constant. A *Boolean formula* is an atom, the complement of a formula, a disjunction of formulas, or a conjunction of formulas. A *literal* as an atom or its complement. A *term* is a conjunction of literals. A *Boolean function* is a function which can be described by a Boolean formula. A *positive* Boolean constraint is of the form $f \subseteq g$ where f and g are Boolean formulas. A *negative* Boolean constraint is of the form $f \not\subseteq g$. A *system* of Boolean constraints is a conjunction of positive and negative Boolean constraints.

Boole showed that any system of positive Boolean constraints can be rewritten to an equivalent Boolean equation of the form $f = \emptyset$. Similarly, any negative Boolean constraint can be rewritten to an equivalent Boolean dis-equation of the form $f \neq \emptyset$. It therefore follows:

Theorem 3.1 Any system of Boolean constraints can be rewritten into an equivalent system of the form:

$$f = \emptyset \wedge g_1 \neq \emptyset \wedge \ldots \wedge g_n \neq \emptyset$$

where f and the g_i's are Boolean formulas.

A fundamental result of Boole is that positive constraints are closed under existential quantification. More precisely, letting $f_x(a)$ denote the formula obtained by replacing all occurrences of x in f by a, we have that:

Theorem 3.2 (Boole)

$$\exists x \,.\, f = \emptyset \Leftrightarrow f_x(\emptyset) \cdot f_x(1) = \emptyset.$$

Unfortunately arbitrary systems of Boolean constraints are *not* closed under existential quantification. To see this, consider the existentially quantified system $\exists x \,.\, x \cdot y \neq \emptyset \wedge \overline{x} \cdot y \neq \emptyset$. This system implies that $|y| \geq 2$, but there is no system of Boolean constraints over y which can capture this.

We are therefore interested in finding the *best approximation* for a quantified system $\exists x \,.\, S$, that is, the maximal system implied by $\exists x \,.\, S$. The best approximation always exists, namely:

$$\bigwedge \{C \mid C \text{ is a constraint s.t. } vars(C) \subseteq vars(S) \\ \wedge \exists x \,.\, S \Rightarrow C\}.$$

Although this is a finite conjunction, computationally it is not a very satisfactory characterization. We now develop a more tractable characterization. This hinges on two results. First, that systems in which there is only one dis-equation are, unlike general systems, closed under existential quantification. Second, that a sort of "weak" independence of negative constraints holds, namely that the best approximation to

$$\exists x \,.\, f = \emptyset \wedge g_1 \neq \emptyset \wedge \ldots \wedge g_n \neq \emptyset$$

is equivalent to

$$\exists x \,.\, (f = \emptyset \wedge g_1 \neq \emptyset) \wedge \ldots \wedge \exists x \,.\, (f = \emptyset \wedge g_n \neq \emptyset).$$

Lemma 3.3 For arbitrary elements a, b, c, d:

$$\begin{aligned} & \exists x \,.\, a \subseteq x \subseteq b \wedge \neg\, (c \subseteq x \subseteq d) \\ \Leftrightarrow \quad & a \subseteq b \wedge \neg\, (b \subseteq d \wedge c \subseteq a) \end{aligned}$$

Proof: "$\Rightarrow$": Clearly, $a \subseteq b$ follows from the antecedent. Assume that the second part of the consequent does not hold, then $b \subseteq d \wedge c \subseteq a$. Together with $a \subseteq b$ this implies $c \subseteq a \subseteq b \subseteq d$, which contradicts the antecedent.
"$\Leftarrow$": If we assume $a \subseteq b \wedge b \not\subseteq d$, the consequent holds with $x = b$. On the other hand, assuming $a \subseteq b \wedge c \not\subseteq a$, the consequent holds with $x = a$. ■

Theorem 3.4 Let S be the system $f = \emptyset \wedge g \neq \emptyset$. Then

$$\begin{aligned} & \exists x \,.\, S \;\Leftrightarrow \\ & f_x(\emptyset) \cdot f_x(1) = \emptyset \wedge \overline{f_x(1)} \cdot g_x(1) + \overline{f_x(\emptyset)} \cdot g_x(\emptyset) \neq \emptyset. \end{aligned}$$

Proof: Let A be $f_x(\emptyset)$, B be $f_x(1)$, C be $g_x(\emptyset)$ and D be $g_x(1)$.

$$\begin{aligned} \exists x \,.\, S \quad \Leftrightarrow \quad & \text{(from Theorem 3.9)} \\ & \exists x \,.\, A \subseteq x \subseteq \overline{B} \wedge \neg\, (C \subseteq x \subseteq \overline{D}) \end{aligned}$$

$$\begin{array}{ll} \Leftrightarrow & \text{(from Lemma 3.3)} \\ & A \subseteq \overline{B} \wedge \neg\,(\overline{B} \subseteq \overline{D} \wedge C \subseteq A) \\ \Leftrightarrow & A \subseteq \overline{B} \wedge (\overline{B} \not\subseteq \overline{D} \vee C \not\subseteq A) \\ \Leftrightarrow & A \cdot B = \emptyset \wedge (\overline{B} \cdot D \neq \emptyset \vee C \cdot \overline{A} \neq \emptyset) \\ \Leftrightarrow & A \cdot B = \emptyset \wedge \overline{B} \cdot D + C \cdot \overline{A} \neq \emptyset. \quad \blacksquare \end{array}$$

Definition. Let S be the system

$$f = \emptyset \wedge g_1 \neq \emptyset \wedge \ldots \wedge g_n \neq \emptyset.$$

Define $proj(S,x)$ to be

$$A{\cdot}B = \emptyset \wedge \overline{B}{\cdot}D_1{+}\overline{A}{\cdot}C_1 \neq \emptyset \wedge \ldots \wedge \overline{B}{\cdot}D_n{+}\overline{A}{\cdot}C_n \neq \emptyset$$

where A is $f_x(\emptyset)$, B is $f_x(1)$, C_i is $(g_i)_x(\emptyset)$ and D_i is $(g_i)_x(1)$.

It follows from Theorem 3.4 that $\exists x\ .\ S$ implies $proj(S,x)$. Moreover, we can show that $proj(S,x)$ is the maximal necessary condition. To show this, we first prove that there is a class of Boolean algebras in which $\exists x . S$ is in fact equivalent to $proj(S,x)$.

Definition. A non-empty element x of a Boolean algebra M is *atomic* iff there exists no element y in M such that $\emptyset \subset y \subset x$. M is *atomless* iff it contains no atomic elements [4].

An example of an atomless Boolean algebra is the set of (equivalence classes of) measurable subsets of $\Re^k$, in which two sets are considered equivalent when they are identical "almost everywhere". This Boolean algebra corresponds to the data model in spatial databases in which regions are not arranged on a grid. An important property of atomless Boolean algebras is that independence of negative constraints holds:

Theorem 3.5 (Independence) For all atomless Boolean algebras M:

$$\begin{array}{lll} M & \models & \exists x\ .\ f = \emptyset \wedge g_1 \neq \emptyset \wedge \ldots \wedge g_n \neq \emptyset \\ & & \Leftrightarrow \\ & & \exists x\ .\ (f = \emptyset \wedge g_1 \neq \emptyset) \wedge \ldots \wedge \\ & & \exists x\ .\ (f = \emptyset \wedge g_n \neq \emptyset). \end{array}$$

Proof: (Sketch) Direction "$\Rightarrow$" is trivial. To show "$\Leftarrow$", assume that $\exists x\ .\ (f = \emptyset \wedge g_i \neq \emptyset)$ holds for $i = 1, \ldots, n$. We can represent functions f and g_i in disjunctive normal form as follows:

$$f = \sum_j x \cdot r_j \ + \ \sum_j \overline{x} \cdot s_j$$

where terms r_j and s_j are nonempty, mutually disjoint, and do not contain variable x. Likewise, we can represent each g_i as:

$$g_i = \sum_j x \cdot u_{i,j} \ + \ \sum_j \overline{x} \cdot v_{i,j}$$

where again $u_{i,j}$ and $v_{i,j}$ are nonempty, mutually disjoint, and do not contain x. Furthermore, we require that every $u_{i,j}$ and $v_{i,j}$ is either equal to some r_j or s_j, or that it is disjoint from every r_j and s_j. That such a representation exists, is a consequence of Theorem 3.10. Since M is atomless, we can find for every $u_{i,j}$ and $v_{i,j}$ a proper, nonempty subset $u'_{i,j}$ and $v'_{i,j}$. Define

$$X = \sum_{i,j} u'_{i,j} + \sum_{i,j} v'_{i,j} + \sum_j s_j - \sum_j r_j$$

Then, using $f = \emptyset$ and $g_i \neq \emptyset$, we can show that $f_x(X) = \emptyset$, and $(g_i)_x(X) \neq \emptyset$, for $i = 1, \ldots, n$. Hence, the left side of the equivalence holds with $x = X$. $\blacksquare$

Thus, atomless boolean Algebras admit quantifier elimination for systems of Boolean constraints:

Theorem 3.6 For all atomless Boolean algebras M:

$$M \models \exists x\ .\ S \Leftrightarrow proj(S,x).$$

Proof: A simple consequence of Theorem 3.4 and Theorem 3.5.

Theorem 3.7 Let S and S' be two systems of Boolean constraints. Then $S \Rightarrow S'$ iff, for all atomless Boolean algebras M, $M \models S \Rightarrow S'$.

Proof: Direction "$\Rightarrow$" is trivial. To show "$\Leftarrow$", assume that $S \Rightarrow S'$ does not hold. Then there is a Boolean algebra M_0 and a substitution σ from variables in S and S' to elements in M_0 such that σS is true and $\sigma S'$ is false. We extend M_0 to an atomless

Boolean algebra by repeatedly executing the following completion step: For every atomic element x of M_0, add two atomic elements x_1 and x_2 to M, such that $x = x_1 + x_2$. Then, $\{x_i + y \mid y \in M_0, i = 1, 2\}$ is a Boolean algebra in which x is non-atomic. Repetition of this step for all atomic x in M_0 gives us a Boolean algebra $T(M_0)$, in which all elements of M_0 are non-atomic. It follows that $\bigcup_{i=1}^{\infty} T^i(M_0)$ is an atomless algebra (call it M) which contains M_0 as a sub-algebra. Since M contains M_0, σS is true and $\sigma S'$ is false in M. Hence, M as well as M_0 is a Boolean algebra of $S \not\Rightarrow S'$. Since M is atomless, the right-hand side of the equivalence cannot hold. ■

Theorem 3.8 $proj(S, x)$ is the best approximation of $\exists x \,.\, S$.

Proof: Let R be another approximation of $\exists x \,.\, S$, such that $\exists x \,.\, S \Rightarrow R$ holds. With Theorem 3.6, we have $M \models proj(S, x) \Rightarrow R$ for all atomless Boolean algebras M. With Theorem 3.7, this implies $M' \models proj(S, x) \Rightarrow R$ for all Boolean algebras M'. ■

Example 3.1 Consider the system S, $x \cdot y \neq \emptyset \wedge \overline{x} \cdot y \neq \emptyset$, from above. In this case $proj(S, x)$ is $y \neq \emptyset$, the best approximation of $\exists x \,.\, S$.

We can iteratively use the function *proj* to compute the strongest Boolean constraint S' which is a necessary condition for $\exists x_{i+1} \,.\, \exists x_{i+2} \,.\, ... \exists x_n \,.\, S$. Using the following two theorems we can transform this system S' into the solved form given in (1).

First, Schröder's theorem that any Boolean equality is equivalent to a range constraint, allows us to rewrite the Boolean equality in S' into a range constraint over x_i.

Theorem 3.9 (Schröder)

$$f = \emptyset \Leftrightarrow f_x(\emptyset) \subseteq x \subseteq \overline{f_x(1)}.$$

Second, Boole's "fundamental theorem of Boolean algebra" allows us to rewrite the Boolean disequations into a form in which x_i is isolated.

Theorem 3.10 (Boole) $f = x \cdot f_x(1) + \overline{x} \cdot f_x(\emptyset)$.

The last three theorems give rise to the algorithm in Figure 2 to compute the triangular solved form of a system of Boolean constraints.

Algorithm 3.1

let S =be a system of Boolean constraints
  in n variables $x_1, ..., x_n$;
let $S_n = S$;
for $i := n$ to 1 do
  let $[\![f = \emptyset \wedge g_1 \neq \emptyset \wedge ... \wedge g_m \neq \emptyset]\!] = S_i$;
  let $C_i = [\![\; f_{x_i}(\emptyset) \subseteq x_i \subseteq \overline{f_{x_i}(1)} \wedge$
    $\bigwedge_k x_i \cdot (g_k)_{x_i}(1) + \overline{x_i} \cdot (g_k)_{x_i}(\emptyset) \neq \emptyset$
    $]\!]$
    where k ranges over all formulas
      $g_1, ..., g_m$ in which x_i occurs;
  let $S_{i-1} = proj(S_i, x_i)$;
then $C_1, ... C_n$ is a triangular solved form of S.

Figure 2: Computing Triangular Form

The algorithm first constructs the constraint C_n in $x_1, ..., x_n$ by rewriting the original system according to Theorem 3.9 and Theorem 3.10. Then, variable x_n is eliminated using Theorem 3.8 and the rewrite-step is applied to the resulting system in $n-1$ variables, yielding constraint C_{n-1} in $x_1, ..., x_{n-1}$. The process continues until all variables are eliminated. Using the results we have established in this section, the following theorem is straightforward to prove:

Theorem 3.11 Every system of Boolean constraints has a triangular solved form, which is computed by Algorithm 3.1.

4 Bounding-Box Approximations

In this section, we investigate how to approximate Boolean constraints in solved form by constraints over bounding boxes. In the following, we restrict our attention to Boolean algebras in which bounding box approximations make sense. We are chiefly interested in the k-dimensional space of real numbers, denoted $\Re^k$ and the k-dimensional space of integers. However, our results hold for any Boolean algebra of the form $B = \wp X^k$ where X is a totally ordered infinite set, and X^k has the standard Cartesian ordering.

The *bounding box* of a region $r \subset X^k$, denoted $\lceil r \rceil$, is the set $\{x \in X^k \mid inf\ r \sqsubseteq x \sqsubseteq sup\ r\}$. For example, in $\Re^2$ the bounding box of a region is the minimal enclosing rectangle which has sides parallel to the axes. Operators on bounding boxes are $\sqcap$ (infimum, equivalent to "$\cdot$"), and $\sqcup$ (supremum). Functions constructed from these operators are called *bounding box functions*. An ordering relation on bounding boxes, $\sqsubseteq$, is given by containment. Note that $\sqcup$ is *not* equivalent to set union. Rather, bounding-box union gives the minimal enclosing rectangle of set union.

Bounding boxes are important because queries involving certain constraints over bounding boxes can be efficiently answered with a spatial database which supports range queries. As shown in [12], for example, a single range query can be used to find those objects x satisfying some given conjunction of *bounding box constraints* of the forms:

- $\lceil x \rceil \sqsubseteq a$,
- $b \sqsubseteq \lceil x \rceil$, and
- $\lceil x \rceil \sqcap c \neq \emptyset$.

where a, b and c are given bounding boxes. This is done by representing rectangles in a X^k as points in space X^{2k} and performing a range query on X^{2k}. Figure 3 shows a combination of three such constraints over intervals on the real line. Each axis of the diagram corresponds to one endpoint of an interval. The shaded rectangle represents the set of intervals $\{x \mid a \sqsubseteq \lceil x \rceil \wedge \lceil x \rceil \sqsubseteq b \wedge \lceil x \rceil \sqcap c \neq \emptyset\}$, with constant intervals a, b, and c.

Recall that a system $C_i(x_1, ..., x_i)$ is in solved form if it is of the form

$$s(x_1, ..., x_{i-1}) \subseteq x_i \subseteq t(x_1, ..., x_{i-1}) \wedge \\ r_1 \neq \emptyset \wedge ... \wedge r_m \neq \emptyset$$

where each r_j has the form

$$x_i \cdot p(x_1, ..., x_{i-1}) + \overline{x_i} \cdot q(x_1, ..., x_{i-1}).$$

We are interested in finding the strongest bounding box constraints over $\lceil x_i \rceil$ which are necessary conditions for C_i to hold.

Figure 3: Range Query

Given that we have retrieved values for $x_1, ..., x_{i-1}$, the best bounding-box constraint approximation to the range constraint

$$s(x_1, ..., x_{i-1}) \subseteq x_i \subseteq t(x_1, ..., x_{i-1})$$

is just

$$\lceil s(x_1, ..., x_{i-1}) \rceil \subseteq \lceil x_i \rceil \subseteq \lceil t(x_1, ..., x_{i-1}) \rceil.$$

The dis-equality constraints are a little harder to approximate. The best approximation to

$$x_i \cdot p(x_1, ..., x_{i-1}) + \overline{x_i} \cdot q(x_1, ..., x_{i-1}) \neq \emptyset$$

is $\lceil x_i \rceil \cdot \lceil p(x_1, ..., x_{i-1}) \rceil \neq \emptyset$ when $\lceil q(x_1, ..., x_{i-1}) \rceil = \emptyset$, and the (trivial) constraint $true$ otherwise.

The problem with this approximation is that it requires the Boolean functions s, t, p, q to be repeatedly evaluated during query execution. However, this may be too expensive since intersections, unions and complements of arbitrary retrieved regions have to be computed. A cheaper alternative is to find, at compile time, bounding-box functions which approximate these Boolean functions. These bounding-box functions are then used instead of the original Boolean functions when evaluating the query. In general, this will be much cheaper because intersections and unions over bounding boxes are relatively cheap to compute. To retain correctness, we must approximate the lower bound function s from "below" and must approximate the upper bound function t and the dis-equality functions p and q from "above".

Definition. Let f be a Boolean function and F a bounding box function. F *approximates* f *from below*, written $F \uparrow f$, if, for all $x_1, \ldots, x_n \in B$:

$$F(\lceil x_1 \rceil, \ldots, \lceil x_n \rceil) \sqsubseteq \lceil f(x_1, \ldots, x_n) \rceil.$$

F *approximates* f *from above*, written $F \downarrow f$, if, for all $x_1, \ldots, x_n \in B$:

$$\lceil f(x_1, \ldots, x_n) \rceil \sqsubseteq F(\lceil x_1 \rceil, \ldots, \lceil x_n \rceil).$$

Note that $F \uparrow f$ is *not* equivalent to $F \sqsubseteq f$. This is because $F \uparrow f$ means only that the result of taking bounding boxes first and then applying F, is contained in the result of applying f first and then taking the bounding box.

Of course, we are interested in finding the best bounding-box approximation to these Boolean functions. Having found these functions, we can substitute these into the original system to obtain the best bounding box constraint approximation.

Definition. The *best* lower and upper bounding-box approximations to a Boolean function f are the bounding box functions L_f and U_f which satisfy:

- $L_f \uparrow f \wedge G \uparrow f \Rightarrow G \sqsubseteq L_f$,
- $U_f \downarrow f \wedge G \downarrow f \Rightarrow G \sqsupseteq U_f$.

That these optimal approximations exist is a simple consequence of the fact that bounding boxes form a complete lattice. The question remains how to find them. A simple syntactic transformation such as replacing all $(+)$ and $(\cdot)$ operators in the Boolean function by $(\sqcup)$ and $(\sqcap)$ is not sufficient to arrive at the best upper or lower bound. In part, this is because two Boolean formulas denoting the same Boolean function may denote different bounding-box functions when the above syntactic transformation is performed on them. For example, although $x \cdot y + x \cdot z = x \cdot (y + z)$, in general $x \sqcap y \sqcup x \sqcap z \neq x \sqcap (y \sqcup z)$.

The main result of this section is that there are effective algorithm for computing the best lower and upper bounding-box approximations to a Boolean formula.

Lemma 4.1 For all Boolean functions f and g,

$$\lceil f \cdot g \rceil \sqsubseteq \lceil f \rceil \sqcap \lceil g \rceil.$$

Lemma 4.2 For all Boolean functions f, g and h,

$$(f \sqcap g) \sqcup (f \sqcap h) \sqsubseteq f \sqcap (g \sqcup h).$$

Lemma 4.3 For all Boolean functions f and g,

$$f \subseteq g \Rightarrow \lceil f \rceil \sqsubseteq \lceil g \rceil.$$

Lemma 4.4 For all Boolean functions f and variables x_i,

$$\lceil x_1 \rceil \sqcap \ldots \sqcap \lceil x_n \rceil \sqsubseteq \lceil f \rceil \;\Rightarrow\; x_1 \subseteq f \vee \ldots \vee x_n \subseteq f.$$

Proof: (Sketch) The proof is by contradiction. Assume $\lceil x_1 \rceil \sqcap \ldots \sqcap \lceil x_n \rceil \sqsubseteq \lceil f \rceil$ and $x_i \not\subseteq f$ for all i. Then there exist terms $s_1, \ldots, s_n$ such that $s_i \subseteq x_i$ and $s_i \cdot f = \emptyset$. Partition X^k into n disjoint sets c_i such that the intersection of their bounding-boxes $\sqcap_i \lceil c_i \rceil$ is non-empty. Consider then the variable assignment σ defined by:

$$\sigma\ z = \bigcup \{c_i \mid z \text{ is a positive literal in } s_i\}$$

Then $\sigma\ s_1 + \ldots + \sigma\ s_n = 1$. Now all s_i are disjoint from f, so $\sigma\ f = \emptyset$, thus $\lceil \sigma\ f \rceil = \emptyset$. On the other hand, we have $\lceil \sigma\ s_1 \rceil \sqcap \ldots \sqcap \lceil \sigma\ s_n \rceil \neq \emptyset$, and therefore, since $s_i \subseteq x_i$, also $\lceil \sigma\ x_1 \rceil \sqcap \ldots \sqcap \lceil \sigma\ x_n \rceil \neq \emptyset$. But this means that $\lceil \sigma\ x_1 \rceil \sqcap \ldots \sqcap \lceil \sigma\ x_n \rceil \not\sqsubseteq \lceil \sigma\ f \rceil$, a contradiction. ∎

We can now give a simple characterization of the best lower bound of a Boolean function. Note that we have proved it for the case when $\emptyset$ and 1 are the only constants in the Boolean function. We conjecture that it holds when arbitrary constants are allowed.

Theorem 4.5 For all Boolean functions f,

$$L_f = \bigsqcup_{atom\ x \subseteq f} \lceil x \rceil.$$

Proof: We first show that L_f must be a disjunction of atoms. Let F be the "disjunctive normal form" of L_f. From Lemma 4.2, $F \sqsubseteq L_f$. If F contains some non-unary conjunction, $\lceil x_1 \rceil \sqcap ... \sqcap \lceil x_n \rceil$ then from Lemma 4.4, for some i, $x_i \subseteq F$, and so $\lceil x_i \rceil \sqcup L_f$ will be a bigger lower bounding-box approximation to f, contradicting that L_f is the best. Thus F must consist only of unary conjunctions, and thus L_f must be a disjunction of atoms. That L_f should be this disjunction of atoms follows from Lemma 4.3 and Lemma 4.4. ■

Lemma 4.6 Let F be a bounding box function, t a term and x a variable. Then,

$$\lceil \overline{x} \cdot t \rceil \sqsubseteq F \Rightarrow \lceil t \rceil \sqsubseteq F.$$

Proof: We must have that $\lceil t \rceil \sqsubseteq F_x(\emptyset)$. As F is monotonic, $\lceil t \rceil \sqsubseteq F$. ■

Lemma 4.7 The best upper bounding box approximation to the conjunction of variables $x_1 \cdot ... \cdot x_n$ is $\lceil x_1 \rceil \sqcap ... \sqcap \lceil x_n \rceil$.

Lemma 4.8 Let f and g be Boolean functions. Then, $U_{f+g} = U_f \sqcup U_g$.

Theorem 4.9 For all Boolean functions f,

$$U_f = \bigsqcup_{term\ t \in SOP(f)} (\sqcap_{atom\ x \in t} \lceil x \rceil\ )$$

where $SOP(f)$ stands for any sum-of-products representation of f.

Proof: It follows from Lemma 4.1, Lemma 4.6 and Lemma 4.7 that $\sqcap_{atom\ x \in t}\ \lceil x \rceil$ is the best upper bounding box approximation to a term t. The result therefore follows from Lemma 4.8. ■

We now present an algorithm for computing L_f and U_f. This algorithm makes use of the Blake canonical form [1], $BCF(f)$ of a Boolean function f, which consists of the sum of all prime implicants of f.

Definition. A *prime implicant* of a function f is a term p such that $p \subseteq f$ and $q \not\subseteq f$ for all true subterms q of p. A sum-of-products formula f is *syllogistically less* than a sum-of-products formula g (in symbols: $f << g$), iff all terms in f have subterms in g.

The following proposition gives the reason why we are interested in Blake-canonical forms and their duals: They allow us to replace semantic inequality by syllogistic inequality, which can be checked syntactically:

Theorem 4.10 (Blake) For all Boolean formulas f and sum-of-products formulas g:

$$g \subseteq f \Leftrightarrow g << BCF(f).$$

There exist a number of algorithms to compute $BCF(f)$. One method [3] first converts f to an arbitrary sum-of-products formula and then repeatedly forms the consensus of two terms in f and simplifies by absorption until a fixpoint is reached. Forming consensus means rewriting according to the rule:

$$x \cdot p + \overline{x} \cdot q \ = \ x \cdot p + \overline{x} \cdot q + p \cdot q.$$

Simplifying by absorption means rewriting according to the rule:

$$p \cdot q + p \ = \ p.$$

Example 4.1 Consider $f = \overline{x} \cdot y + x \cdot (y + z \cdot \overline{w})$. BCF(f) is computed as follows:

$$\begin{array}{rcll} f & = & \overline{x} \cdot y + x \cdot y + x \cdot z \cdot \overline{w} & \text{(Distribution).} \\ & = & y + \overline{x} \cdot y + x \cdot y + x \cdot z \cdot \overline{w} & \text{(Consensus)} \\ & = & y + x \cdot y + x \cdot z \cdot \overline{w} & \text{(Absorption)} \\ & = & y + x \cdot z \cdot \overline{w} & \text{(Absorption)} \end{array}$$

Theorem 4.10 shows that an atom is stronger than a formula f iff it occurs as an atom in $BCF(f)$. This observation gives rise to the following algorithm for computing L_f and U_f:

Algorithm 4.1

Given a Boolean function f,

1. compute $BCF(f)$,
2. take the bounding-box union of all terms in $BCF(f)$ consisting of single atoms, yielding L_f,

3. remove all negative literals from $BCF(f)$, and then simplify, yielding U_f.

Theorem 4.11 Algorithm 4.1 computes the best lower an upper bounding-box approximations to a given Boolean function f.

Proof: Immediate from Theorem 4.5, Theorem 4.9 and Theorem 4.10. ■

Example 4.2 Consider function f from Example 4.1. We have seen that $BCF(f) = y + x \cdot z \cdot \overline{w}$. Since the only atom in $BCF(f)$ is y, we have $L_f = y$. U_f is computed by dropping all negative literals of f and converting to bounding box form, yielding $U_f = y \sqcup (x \sqcap z)$.

The time to compute BCF is exponential in the number of variables in the formula. Hence, Algorithm 4.1 has (just as Algorithm 3.1) complexity exponential in the number of variables in in the solved constraint system. We feel that in practice this will not be a problem, since both algorithms are executed during query compilation rather than in query evaluation, and the number of variables in a constraint system can be expected to be reasonably small.

5 Conclusions

We have presented a method to approximate a system of multivariate Boolean constraints by a sequence of univariate range queries. The method is a special case of a more general optimization which maps constraints in several variables into database searches for individual objects. It can be extended to other application areas, such as constraints over the real closed fields. The optimization relies on converting a given system into triangular form. The triangular form is obtained by repeatedly computing the existential quantification $\exists x.C$ of a constraint system C. If the constraint language is not closed under existential quantification, an approximation may be used.

References

[1] A. Blake. *Canonical Expressions in Boolean Algebra.* PhD thesis, Uni. of Chicago, 1937.

[2] G. Boole. *An Investigation of the Laws of Thought.* Walton, London, 1847. (Reprinted by Philisophical Library, New York, 1954).

[3] F.M. Brown. *Boolean Reasoning, The Logic of Boolean Equations.* Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1990.

[4] C.C. Chang and H.J. Keisler. *Model Theory, 2nd Edition.* North-Holland, 1977.

[5] N.S. Chang and K.S. Fu. Picture query languages for pictorial database systems. *IEEE Computer Magazine*, 14(11):23–33, 1981.

[6] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *ACM/SIGMOD Annual Conference on Management of Data*, pages 47–57, Boston, MA, 1984.

[7] R. Helm, K. Marriott, and M. Odersky. Building visual language parsers. In *Computer Human Interaction (CHI)*, pages 105–112. ACM Press, 1991.

[8] T. Joseph and A. F. Cardenas. PICQUERY: a high level query language for pictorial database management. *IEEE Transactions on Software Engineering*, 14(5):630–638, May 1988.

[9] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: an adaptable, symmetric multi-key file structure. *ACM Trans. on Database Systems*, 9(1):38–71, 1984.

[10] J.A. Orenstein and F.A. Manola. PROBE spatial data modelling and query processing in an image database application. *IEEE Trans. on Software Eng.*, 14(5):611–629, May 1988.

[11] S. Rudeanu. *Boolean Functions and Equations.* Elsevier Science Publishers B.V.(North Holland), New York, N.Y., 1974.

[12] H. Samet. Hierarchical representations of collections of small rectangles. *ACM Computing Surveys*, 20(4):271–309, December 1988.

[13] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: a dynamic index for multi-dimensional objects. In *Very Large Databases (VLDB)*, 1987.

[14] J.D. Ullman. *Computational aspects of VLSI*. Computer Science Press, Rockville, Maryland, 1984.