Constraint Database Query Evaluation with Approximation

Pierangelo Veltri
INRIA-Rocquencourt, BP 105, F-78153 Le Chesnay, France
Pierangelo.Veltri@inria.fr

Abstract

We consider the problem of solving a large number of simple systems of linear constraints. This problem occurs in the context of constraint databases. The developed methodology is based on a hierarchical evaluation of the constraints which are first simplified, and replaced by approximations.

We focus on systems of linear constraints over the reals, which model spatial objects, and consider both geometric and topological approximations, defined with very simple constraints. We show that these constraints can be used either to solve the initial systems, or at least to filter out unsatisfiable systems. The main contribution of the paper is the development of a set of rewriting rules, that allow the transformation of spatial object queries into equivalent ones that make use of the approximation, reducing the query evaluation cost.

1. Introduction

Constraints are used in various areas of computer science, such as for instance logic programming, combinatorial optimization, images animation, spatio-temporal database ([11], [12], [7], [5]). In this paper, we consider the problem of solving a *large* number of *simple* systems of constraints, which occurs in the context of databases. The emphasis is therefore on the number of systems we want to solve, versus the complexity of each individual system. The problem is motivated by applications such as those involving spatial data, where an efficient data manipulation is strongly required.

Spatial databases contain collections of geometric objects, such as polygons in a 2d space. The manipulation of these objects is costly because of the intricacy of the geometry. To guaranty efficient access to the geometric objects in a database, special data structures (indexes) are designed, which ensure a logarithmic time search of objects. In the case of spatial data, the logarithmic time cannot be guaranteed, but can be reached in practice through spatial indexes such as the R-tree [10] and its variants [15, 1]. Spatial indexes are not designed directly on the geometry of the objects, but on their *minimal bounding boxes*, that is the smallest rectangular polyhedron containing it. The minimal bounding box constitutes an object approximation that can be used to help geometric processing.

Geometric objects as polygons can be easily represented by systems of linear inequalities over the reals. See Fig. 1 for an example. The minimal bounding boxes on the other hand can be represented by systems of univariate constraints, which are of course much easier to process than the general linear constraints. The use of constraints to represent complex data is also advocated in [9], [8], [6].

$$\begin{aligned} & y \leq x && (1)\\ \wedge\ & x \leq 5 && (2)\\ \wedge\ & y \geq 1 && (3)\\ \wedge\ & 3y + x \leq 24 && (4) \end{aligned}$$

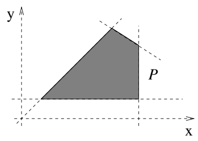

Figure 1. A system of constraints and its associated geometry $\mathcal{P}$

We develop a strategy to solve the systems of constraints, which is based on a hierarchical evaluation. We first consider a geometric approximation using simplified constraints. The evaluation over simplified constraints can be done quite efficiently, and allows to determine in many cases whether the initial system is satisfiable or not. For the data that qualify to this *filtering step*, the initial constraints need generally to be processed. In practice, the process allows to reduce the application of the costly constraint solving operations to a small subset of the input, and results in drastic optimization.

In the present work, we consider a hierarchical evaluation technique, based on three classes of constraints, (i) *linear constraints* over the reals, (ii) *box constraints*, which are univariate constraints, and (iii) *topological constraints*.

We assume that the objects in the database are stored with both a *geometry*, which is a polyhedron represented using linear constraints, and a *bounding box*, which is a rectangular approximation of the geometry. Observe that the bounding-box approximation is *conservative*: any point in the geometric object belongs to the bounding-box as well.

Moreover we assume that three of the Egenhofer and Franzosa [3] topological relations between geometric objects are stored in the database: *overlap, disjoint*, and *inside* (see Fig. 3). We assume that the database contains these binary relations, which store pairs of database objects that satisfy the corresponding topological condition.

In databases, objects are manipulated through query languages. The basic operations require the processing of constraints to produce an output. The goal of this paper is to show how the box constraints and the topological constraints can be involved in an efficient query processing of general linear constraints. Our contribution is the design of rewriting rules for algebraic queries. We prove that the rewriting system is sound and optimal for the basic operations of relational algebra.

The paper is organized as follows. In the following section, we formally introduce constraint approximations. In Section 3, we define the model used to represent geometric objects and a set of relational algebra operations to manipulate them. In Section 4 we present a manipulation language for collections of spatial data. The rewriting rules for constraint processing are presented in Section 6. We then conclude presenting a practical example.

2. Constraint approximations

We consider linear constraints over the reals of the form:

$$\sum_{i=1}^{p} a_i\, x_i = a_0, \qquad \sum_{i=1}^{p} a_i\, x_i \leq a_0$$

where the x_i's denote variables and the a_i's are rational constants, and we don't consider irrational parameters. Fig. 1 shows for instance a system of constraints and its corresponding polygon $\mathcal{P}$.

We study systems of linear constraints over d variables for some fixed d which corresponds to the dimension of the space. To catch the intuition, we fix $d = 2$ in examples. Each such system defines a convex polyhedron in the real space.

We consider the manipulation of these polyhedra through various operations, such as geometric transformations: e.g.,projection, and set-theoretic operations: e.g.,intersection. These operations are evaluated by processing the constraints defining the objects. The complexity of their evaluation strongly depends upon the number of constraints involved, and upon the global dimension ([4]).

If the operations need to be applied to large collections of objects, the constraint processing phase should be optimized. Our approach consists in considering an approximated representation of the objects with simpler constraints that can be evaluated efficiently. We consider box constraints and topological constraints.

Definition 1 A *box constraint* is a linear constraint with a unique variable, of the form $x = a$, or $x \leq a$, for a variable x and a constant a.

The resolution of systems of box constraints can be done in time linear in the number of constraints independently of the dimension.

We define a *bounding box approximation (box)* of a system of linear constraints representing a polyhedron as the system of box constraints representing the minimal bounding box of the polyhedron. The bounding box approximation can be used to increase the efficiency of linear constraint processing. Consider for instance the projection π of the polygon $\mathcal{P}$ of Fig. 1 on either x or y. The bounding box approximation of $\mathcal{P}$ is given by $1 \leq x \leq 5 \wedge 1 \leq y \leq 6$. Thus the projection of $\mathcal{P}$ on the x-axis can be obtained directly from the approximation:

$$\pi_x \mathcal{P} = \pi_x box(\mathcal{P})$$

More in general, the projection is obtained eliminating variables from the system of constraints. The cost of eliminating *one* variable from a system of n constraints over d variables is linear for $d = 1$, in $O(n \log n)$ for $d = 2$, and quadratic in general using the Fourier method [14].

Consider now the intersection operation, $\cap$. Two objects have a non empty intersection only if their bounding box approximations have a non empty intersection. For instance consider the Fig. 2. With the bounding box approximation, one of the two following cases occurs: (i) the minimal bounding box of the two objects intersect, the two objects are candidates and the value needs to be computed (Fig. 2.b and Fig. 2.c), or (ii) the minimal bounding box of the two objects do not intersect, and the result (empty) is obtained with no computation (Fig. 2.a). We observe that if object approximations intersect the result can nevertheless be empty (Fig. 2.b).

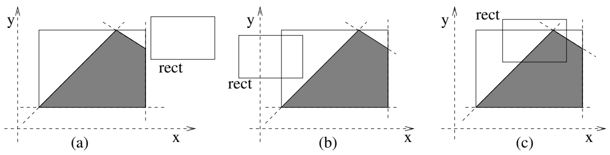

Figure 2. Intersection with a rectangle

The intersection of two objects defined using linear constraints can be defined by the conjunction of their systems of constraints. From a geometric point of view, the computation corresponds in a simplification of the resulting system. Its complexity increases with the number of variables. In dimension 2 it can be performed in $O(n \log n)$ by a half-plane intersection algorithm [13]. On the other hand the complexity of evaluating the intersection between two bounding box approximations is linear.

Let us consider now how the topology can be used to increase the efficiency of linear constraint processing. We consider objects o which have identifiers o_id, a geometry denoted by $o.geo$, and a bounding box denoted by $o.box$ defined with linear constraints.

Definition 2 A *topological constraint* is one of the following expressions: *disjoint(o,o'), inside(o,o'), overlap(o,o')* where *o* and *o'* are objects.

Definition 3 Let o and o' be objects with geometry $o.geo$, and $o'.geo$. The topological predicates *disjoint*, *inside*, *overlap*, are defined as follows: (i) *disjoint(o,o')* is true if and only if $o.geo \cap o'.geo = \emptyset$. (ii) *inside(o,o')* is true if and only if $o.geo \cap o'.geo = o.geo$. (iii) *overlap(o,o')* is true if and only if $o.geo \cap o'.geo \neq \emptyset$ and $o.geo \cap o'.geo \neq o.geo$ and $o.geo \cap o'.geo \neq o'.geo$.

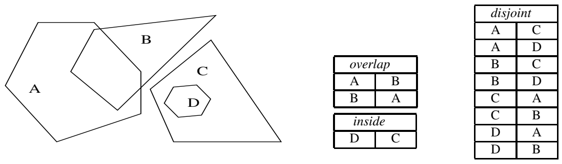

Figure 3. Geometric objects and topology

For the intersection between two objects *o* and *o'*, using the topological approximation one of the following cases occurs: (i) *disjoint(o,o')*: the result (empty) can be obtained directly from the topological relation; (ii) *overlap(o,o')*: the pair of objects is accepted and intersection value needs to be computed; (iii) *inside(o,o')*: the object is accepted and the result,*o*, is obtained with no computation.

3. Geometric data modeling

Geometric objects are modeled as infinite sets of points in the real space. For example, if $\mathbb{R}$ is the universe of the real number, a polygon in the plane is seen as the infinite set of points of $\mathbb{R}^2$ inside its frontier.

At this *abstract level*, spatial data consists of infinite relations over the real numbers $\mathbb{R}$. These infinite relations can only be described and manipulated through a finite representation.

Following the trends of constraint databases [11], we use first-order logic to represent the relations of interest, and define a *symbolic level* of representation. We distinguish clearly between the *intensional* representation of a relation at the symbolic level, and its *extensional* interpretation at the abstract level.

The query evaluation is based on an *algebra*, equivalent to first-order logic, consisting of the following operations: Cartesian product, $\times$, projection, π, union, $\cup$, set difference, $-$. These operations, whose effect is described below, are performed on sets of symbolic tuples where each symbolic tuple is a conjunction of constraints representing a geometric object. Let R_1 and R_2 be two relations, and respectively e_1 and e_2 be sets of symbolic tuples defining them.

1. $R_1 \times R_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in e_2\}$.
2. $\sigma_F(R_1) = \{t_1 \wedge F \mid t_1 \in e_1\}$ where F is a condition expressed by a set of conjunction of constraints.
3. $\pi_{\overline{x}}\ R_1$ is computed using the algorithm of quantifier elimination[1].
4. $R_1 \cup R_2 = e_1 \cup e_2$.
5. $R_1 - R_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in (e_2)^c\}$, where e^c is the set of tuples corresponding to $\neg e$
6. $simplify(R_1)$, eliminates redundancies and detects inconsistencies.

The symbolic operations above are well defined in the sense that their effect on the intensional definition of sets corresponds exactly to the semantic of the corresponding relational operators from the relational algebra over the possibly infinite extension of the sets. The relational intersection and join are definable with the symbolic Cartesian product. The *simplify* operator, given a set of conjunction of constraints, eliminates redundancies, and detects inconsistencies.

4. Collections data modeling

We consider *data collections* as a collection of real-world entities which includes: (i) geometric objects with their *spatial description* and (ii) an abstract description of such objects. Practical examples can be found in geographic applications, where an abstract description of the real-world distinguishes different classes of entities on the earth surface: *buildings*, *roads*, *parcels*. Since applications aim at manipulating the location of these objects, we need a data model allowing the representation and manipulation of classical attributes (*owner* for instance) as well as geometric attributes.

To this end we store data in non-first normal form relations [16] with one level of set nesting to represent geometric attributes as first class citizens. Moreover geometric objects are seen as infinite set of points and represented in finite way using constraints. This model is the one used in DEDALE [4] and advocated in [2].

Fig. 4 shows an example of a collection schema describing the class *Parcel* and its instance: *owner* and *ground-occupancy* are represented as classical atomic values, while

[1] In the linear case it is done by the Fourier-Motzkin elimination method [14].

$$Parcel = \quad \{[\quad \text{owner} : string$$
$$\text{ground-occupancy (go)} : string$$
$$\text{geo} : geometry\ ]\}$$

owner	go	geo
John	Pasture	$(1230x + 134y \leq 23$ $\wedge(3x - 2y \leq 49 \wedge 23x + 2y \geq 134$ $\ldots \wedge 3x - 467y \geq 265)\ldots$

Figure 4. The Parcel collection

the geometric attributes *geo* are represented using a nested symbolic relation.

5. Querying spatial data collections

Collections of spatial data can be manipulated with relational algebra operators. Moreover, in order to manipulate and query geometric data, we introduce a Map operator to apply algebraic expressions to nested spatial attributes. Given a collection R (as that of Fig. 4), $Map_{\lambda X.[E(X)]}\, R$ defines a collection of objects R', in which each object o of R has been replaced by $E(o)$, and $E(X)$ is a tuple of algebraic expressions, $E(X) = e_1(x), ...e_m(x)$.

As a simple example, consider the following *clipping query* over the collection $Parcel$: *parts of parcels which cross the rectangle rect* Assuming that *rect* is represented by the constraints $\{x_{min} \leq x \leq x_{max} \wedge y_{min} \leq y \leq y_{max}\}$, the logical expression is

$$Map_{\lambda X.[X.owner, X.geo \cap rect]}(Parcel)$$

The semantic of $X.geo \cap rect$ is the pairs (x, y) in the *geo* attribute of $Parcel$ such that (x, y) can be also found in *rect*. Since both extensions are infinite, this result is represented by linear constraints. Those objects of R whose $X.geo \cap rect = \emptyset$ are discarded. Upon the instance of Fig. 4, this yields the collection of Fig. 5.

In most cases the evaluation of geometric queries can be done in two steps. The first one is purely symbolic: we just make some manipulation of the constraint representation (a trivial conjunction in the example above). In a second step it remains to test for emptiness the resulting formula which might be unsatisfiable. If we consider that in each collection there is a minimal bounding box attribute associated to each geometric value, the query can be rewritten as:

$$Map_{\lambda X.[X.owner, rect \cap X.geo]}(\sigma_{rect \cap box \neq \emptyset}(Parcel))$$

By testing the overlapping of *rect* and *box*, we can eliminate from further consideration a large amount of parcel entities for which the geometric computation would yield an empty result. This is a simple but representative example of query optimization based on bounding box approximation.

owner	geo
John	$(1230x + 134y \leq 23$ $\wedge(3x - 2y \leq 49 \wedge 23x + 2y \geq 134\ldots$ $\wedge 3x - 467y \geq 265)$ $\wedge x_{min} \leq x \leq x_{max} \wedge y_{min} \leq y \leq y_{max})\ldots$

Figure 5. The result of a clipping query

6. Query rewriting

In this section we introduce some rewriting rules for geometric operations based on the use of topological and bounding box constraints. Such operations are very frequent in spatial queries. We deal with two sample collections R and R' containing objects stored with the geometry (*geo*) and its bounding box approximation (*box*). Moreover we assume that topological relations (*overlap*, *disjoint*, and *inside*) between objects identifiers (o_id) are stored in the database. The proofs of the following lemmas that can be found in the full paper [17].

- Intersection operation, $\cap$, between a fixed rectangle *rect* and the geometric attribute of R

Lemma 1

$$Map_{\lambda X.[X.geo \cap rect]}\, R \equiv Map_{\lambda X.[X.geo \cap rect]}\, \sigma_{box \cap rect \neq \emptyset} R$$

This query cannot be rewritten using topological relations since *rect* is given by the user, and thus doesn't occur in the topological relations stored in the database.

- Intersection between the geometric attributes of R and R'

Lemma 2

$$Map_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\, R \times R' \equiv$$
$$Map_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\, \sigma_{box_R \cap box_{R'} \neq \emptyset} R \times R'$$

Using the topological relations the above operation can then be evaluated as:

$$Map_{\lambda X.[X.geo_R \cap X.geo_{R'}]}\, R \bowtie_{o\_id=\$1} overlap \bowtie_{\$2=o\_id} R'$$
$$\cup\, Map_{\lambda X.[X.geo_R]}\, R \bowtie_{o\_id=\$1} inside \bowtie_{\$2=o\_id} R'$$
$$\cup\, Map_{\lambda X.[X.geo_{R'}]}\, R' \bowtie_{o\_id=\$1} inside \bowtie_{\$2=o\_id} R$$

where the expression considers the cases *overlap(o,o')*, *inside(o,o')*, and *inside(o',o)*.

- Difference, $-$, between the geometric part of R and a fixed rectangle *rect*

Lemma 3

$$Map_{\lambda X.[X.geo - rect]}\, R \equiv Map_{\lambda X.[X.geo]}\, \sigma_{box \cap rect = \emptyset} R$$
$$\cup \quad Map_{\lambda X.[X.geo - rect]}\, \sigma_{box \cap rect \neq \emptyset} R$$

- Difference between the geometric part of R and R'

Lemma 4

$$Map_{\lambda X.[X.geo_R - X.geo_{R'}]}\, R \times R' \equiv$$
$$Map_{\lambda X.[X.geo_R - X.geo_{R'}]}\, \sigma_{box_R \cap box_{R'} \neq \emptyset}(R \times R')$$
$$\cup \quad Map_{\lambda X.[X.geo_R]}\, \sigma_{box_R \cap box_{R'} = \emptyset}(R \times R')$$

Considering topological relations, for each pairs of objects $o \in R$ and $o' \in R'$ one of the following case occurs: (i) *disjoint(o,o')*: the result o is obtained with no computation; (ii) *inside(o,o')*: o is entirely contained in o', and the result is empty and obtained with no computation; (iii) *inside(o',o)*, *overlap(o,o')*: the result needs to be evaluated on the exact geometric representation. Then the difference operation can be evaluated as:

$$Map_{\lambda X.[X.geo_R]}\, R \bowtie_{o\_id=\$1} disjoint \bowtie_{\$2=o\_id} R' \cup$$
$$Map_{\lambda X.[X.geo_R - X.geo_{R'}]}\, R \bowtie_{o\_id=\$1} overlap \bowtie_{\$2=o\_id} R' \cup$$
$$Map_{\lambda X.[X.geo_R - X.geo_{R'}]}\, R' \bowtie_{o\_id=\$1} inside \bowtie_{\$2=o\_id} R$$

• Projection, π, on one dimension.

Lemma 5 If x_i is an attribute of box, then

$$Map_{\lambda X.[\pi_{x_i} X.geo]}\, R \equiv Map_{\lambda X.[\pi_{x_i} X.box]}\, R$$

• Selection, σ, of objects containing a point $P(a_1, .., a_k)$

Lemma 6

$$Map_{\lambda X.[\sigma_{x_1=a_1,..,x_k=a_k} X.geo]}\, R \equiv$$
$$Map_{\lambda X.[\sigma_{x_1=a_1,..,x_k=a_k} X.geo]}\, \sigma_{P \in box} R$$

• Select the objects of R, that satisfy the constraint F.

Lemma 7

$$Map_{\lambda X.[\sigma_F X.geo]}\, R \equiv$$
$$Map_{\lambda X.[\sigma_F X.geo_A]}\underbrace{(Map_{\lambda X.[X.geo, \sigma_F X.box]}\, R)}_{A}$$

• Join operation, $\bowtie$, between two collections R and R' (overlap join).

Lemma 8

$$Map_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\, R \times R' \equiv$$
$$Map_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\, \sigma_{box_R \cap box_{R'} \neq \emptyset} R \times R'$$

Using the topological relation *overlap*:

$$Map_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\, R \times R' \equiv$$
$$Map_{\lambda X.[X.geo_R \bowtie X.geo_{R'}]}\, R \bowtie_{o\_id=\$1} overlap \bowtie_{\$2=o\_id} R'$$

Theorem The rewriting rules presented in lemmas 1 to 8 are sound and optimal.

The soundness of the rules follows from the proofs of lemmas 1 to 8 ([17]). The optimality is in the sense that no more filtering could have been done using the approximation. Examples of query rewriting, can be found in the full paper [17]. The present techniques are under implementation in the DEDALE system [4], a prototype of constraint spatial database.

Acknowledgments

My deepest thanks to Stéphane Grumbach and Philippe Rigaux: without them this paper would not exist.

References

[1] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*tree : An Efficient and Robust Access Method for Points and Rectangles. In *ACM SIGMOD*, pages 322–331, 1990.

[2] A. Belussi, E. Bertino, and B. Catania. An extended algebra for constraint databases. In *Transaction on Knowledge and Data Engineering, vol.10(5)*. IEEE, 1998.

[3] M. J. Egenhofer and R. Franzosa. Point-set topological spatial relations. *Int. J. Geo. Info. Sys.*, 5(2), 1991.

[4] S. Grumbach, P. Rigaux, and L. Segoufin. The Dedale System for Complex Spatial Queries. In *ACM SIGMOD*, 1998. http://www-rocq.inria.fr/verso/dedale/ .

[5] S. Grumbach, P. Rigaux, and L. Segoufin. Spatio-temporal data handling with constraints. In *ACM-GIS 98*, 1998.

[6] S. Grumbach, P. Rigaux, and L. Segoufin. On the orthographic dimension of constraint databases. In *ICDT, Jerusalem*. Springer Verlag, 1999.

[7] S. Grumbach, P. Rigaux, and L. Segoufin. Manipulating interpolated data is easier than you thought. In *VLDB*, 2000.

[8] S. Grumbach and J. Su. *Queries with arithmetical constraints*. Invited to a special issue, 1997.

[9] S. Grumbach, J. Su, and C. Tollu. *Linear constraint query languages: Expressive power and complexity*. LNCS 960. Springer-Verlag, 1994.

[10] A. Guttman. R-trees : A Dynamic Index Structure for Spatial Searching. In *ACM SIGMOD*, 1984.

[11] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. In *PODS*, Nashville, 1990.

[12] G. Kuper, L. Libkin, and J. Paredaens. *Constraints Databases*. Springer, 2000.

[13] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, 1985.

[14] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[15] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+Tree: A Dynamic Index for Multi-Dimensional Objects. In *VLDB*, 1987.

[16] J. Ullman. *Database and Knowledge Base Systems*. Computer Science Press, 1988.

[17] P. Veltri. Constraint database query evaluation with approximation. In *Manuscript*, 2000. www-rocq.inria.fr/~veltri/papers.html.