Notes on Query Evaluation and Optimization

Stéphane Grumbach Philippe Rigaux Luc Segoufin

December 20, 1999

The optimization of queries over constraint databases is a prerequisite to the use of this methodology for data intensive applications. In this paper, we present some preliminary results, together with promising directions of research to address this fundamental issue.

Although the issue of query optimization and evaluation is relatively new in constraint databases, there exists a large amount of work devoted to spatial query processing in the more general realm of spatial databases. In particular *spatial indices* which are the fundamental tools for supporting query optimization are now well established. The research on query processing, and above all on *spatial join processing* is very active. Most of the work is based on an experimental approach, but it provides already a sound support which should be exploited in the context of constraint databases.

One of the main difficulties of the extension of DBMSs with spatial functionalities is the specification of optimization rules to be integrated in the DBMS optimizer. Since there does not exist a commonly accepted spatial language, the introduction of spatial query processing techniques relies on the existence of extensible optimizers and also strongly depends on the formalism adopted to specify the language. With its sound foundations, the constraint database paradigm offers a promising framework for integrating spatial query optimization principles with classical techniques.

Unfortunately classical constraint solving techniques do not apply in the context of constraint databases. Indeed, most algorithms are concerned with finding *one* solution satisfying a constraint system versus finding *all* solutions. This latter situation is the one of interest for constraint databases.

We adopt the following approach. First, we develop (Section 1) a brief comparison of spatial data modeling principles and query processing techniques in both the contexts of spatial and constraint databases. We show how available techniques developed for spatial databases are relevant and can be used in the context of constraint databases. We then focus on aspects of the optimization which are specific to constraints, and consider the potential for optimization regarding two orthogonal issues: (i) the format of the input data, and its organization (Section 2), and (ii) the type of queries, and their transformation (Section 3).

1 Spatial query processing and optimization

We briefly recall the fundamental principles of spatial data management, and show that they are compatible with a data representation based on constraints. We discuss the modeling of the data, the query languages, the processing of queries, and the optimization. Spatial indices are not addressed in this section.

1.1 Data modeling

The traditional approach for the management of spatial data relies on the customization of a DBMS by the addition of abstract data types (ADTs). These types provide both the support for data representation (hidden to the user) and a set of geometric functions which can be used in the query language.

It has been widely advocated that the geometric spatial data represented in spatial databases by collections of polygons as lists of their vertices in vector mode, could conveniently be represented by linear constraints over rational coefficients, representing the edges of the polygons. Semantically, both representations are equivalent for compact objects. However, the constraint representation of spatial data is not unique and has a significant impact on the cost of the performed operations, as addressed in Section 2.

Constraint relations do not appropriately model spatial data in general. Indeed, in the constraint framework, relations are flat when one of the fundamental features of spatial data, is the ability to associate spatial attributes (an instance of an ADT) to objects. The constraint model can be extended with nesting to model this relationship between an object and and its spatial component. In a nested data model, relations are no longer in disjunctive normal form: tuples are built from atomic attributes and (nested) relations. We illustrate the nesting of relations with Example 1.

Example 1 *Let PEOPLE be a relation representing objects in motion. Attributes name and category are represented as classical atomic string values, while the nested attribute trajectory is an infinite ternary relation over $\mathbb{Q}$ representing its trajectory.*

$$\begin{array}{rll} PEOPLE = & \{[& name : string, \\ & & category : string, \\ & & trajectory : \{\mathbb{Q}^3\} \\ &]\} & \end{array}$$

An instance of a tuple of relation PEOPLE is given in Figure 1. The attributes name and category are represented as equality constraints over the uninterpreted domain string, when the nested attribute trajectory is represented as a first-order formula without quantifiers in DNF. For the sake of simplicity, instead of equality constraints, only string values are given for attributes name and category.

One can allow a recursive application of nesting, but one level seems to suffice to represent spatial data. Essentially the alphanumerical information is stored in the (top

name	category	trajectory
Alice	Tourist	$(x = 1230 \wedge y = 134 \wedge 1 < t < 11)$ $\vee$ $(3x - 2y \leqslant 49 \wedge 23x + 2y \geqslant 134 \wedge 11 < t < 16)$ $\vee$ $\ldots$

Figure 1: An object with nested attribute representing pointsets

level) relation, while the spatial component (a list of points) is nested. This allows in particular to map classical evaluation techniques to constraint databases, as shown in the sequel.

Throughout the remainder of this paper, we assume a data model allowing one level of nesting. Top level relations are finite. Nested relations can be constraint (infinite) relations.

1.2 Query languages

Query languages for spatial data often consists of toolboxes with numerous functions generally oriented towards specific (geometric, topologic, etc.) applications. The use of ADTs often renders the primitive operations rather complex to combine in queries. The greatest benefit of the constraint representation is the possibility to use set-based query languages for manipulating spatial data.

First-order logic, and equivalently relational algebra, can be used to express queries over spatial data represented as constraint relations. Whenever an algebraic operator is applied to (nested) relations, its semantics is defined in a symbolic way as a manipulation of the formulas representing the constraint tuples. The classical algebraic operations can be expressed with only four symbolic operations on sets of constraint tuples. Let R_1 and R_2 be two relations respectively defined by the sets of constraint tuples e_1 and e_2. The four symbolic operators are defined as follows.

1. $R_1 \times R_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in e_2\}$.

2. $\pi_{\bar{x}}\ R_1$ is computed using the algorithm of quantifier elimination, based on Fourier-Motzkin elimination method.

3. $R_1 \cup R_2 = e_1 \cup e_2$.

4. $R_1 - R_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in (e_2)^c\}$, where e^c is the set of tuples or disjuncts of a DNF formula corresponding to $\neg e$.

The symbolic operations above are well defined in the sense that their effect on the intensional definition of sets corresponds exactly to the semantics of the corresponding relational operators from the relational algebra over the possibly infinite extension of the sets. Relational intersection, join and selection are definable with the symbolic Cartesian product. The sets of variables are different in these three operations. For the Cartesian product, the variables of the two relations are disjoint, they are similar in the case of the intersection, and have a non empty intersection in the case of the join and the selection.

Note that normal forms other than DNF can be considered as well. As addressed in Section 2.1, other normal forms can actually represent spatial data in a more effective way than DNF.

Queries over finite (top level) or infinite (nested level) relations can be expressed in the same language. New operators such as MAP, *nest*, *unnest*, *unionnest* and *internest* perform on the one level of nesting. MAP is a restructuring operator that applies to top level relations: if $E(X)$ is an algebraic expression of tuple type T', with a tuple variable $X : T$, and R is a relation of type $\{T\}$ (set of T), then $MAP_{\lambda X.E(X)}(R)$ defines the relation of type $\{T'\}$, in which each tuple t of R, has been replaced by $E(t)$. Operations *nest*, *unnest*, or *unionest* and *internest* that respectively perform a union or an intersection of an attribute are very useful to express spatial queries such as nearest neighbor queries, for instance.

In the last section of this paper, we shall further extend this language with some operations which are not first-order definable over linear signature. They include topological properties, such as connectivity, distances, and geometric constructions, such as the line going through two given points. This gives more expressiveness to the language and allows in particular to construct some fundamental geometric structures such as a convex hull or a Voronoi diagram.

1.3 Query processing

Unlike classical query processing algorithms which make use of a few simple predicates (mostly the equality between two atomic values), the manipulation of spatial data involves a broad range of time-consuming operations (including *topological relationships*, such as *intersection*, and *containment*, *distances*, etc.). The cost of evaluating queries, which is traditionally related to the number of I/O's, becomes quite CPU-bound.

In order to speed-up the repeated application of geometric operations over large collections of data, a spatial query processor involves operators upon an *approximate geometry* (usually the minimal bounding box, *mbb*) and delivers a faster, but approximate answer to queries. Some further steps are then necessary to get the exact answer. The general framework, adopted in virtually all the proposals, is summarized as follows:

1. *Filter step*, the *mbb* is used as an approximation to compute an approximate answer, often called the candidate set. This step can usually be efficiently supported by spatial access methods.

2. *Refinement step*, the members of the candidate set are examined in order to test their exact representation against the query predicates. This exact representation usually consists of a sequence of points in a 2-d space (in vector mode), well suited to the

application of computational geometry algorithms which implement the operations on the ADTs.

The two step evaluation process presented above can easily be implemented for constraints. The mbbs can be explicitly represented as a nested spatial attribute. The filter step can be performed on this latter attribute while the refinement step is performed on the nested finitely representable relation. Queries are then simply expressed in a nested relational algebra, versus functions on the ADTs.

We show in the sequel how the optimization of the filter step, given a constraint algebra ALG can easily be generalized. First we define ALG_{mbb} as the set of algebraic operations restricted to operate only on the bounding box of constraint relations. Given a query $\mathcal{Q}$ expressed ALG by q, we denote by q_{mbb} is the same algebraic expression but applied on the bounding boxes of its input instead of the exact geometry. Clearly, for any conjunctive (hence monotonic) query q, $q(R) \subset q_{mbb}(R)$ for any relation R. The expression $q_{mbb}(R)$ can be seen as the specification of the filter step: it describes the operations to apply upon the relations storing the bounding-box of nested attributes. Any usual rewriting rule may be applied, in conjunction with classical techniques for pruning the search space. For instance the dynamic programming technique proposed in System R can be used to determine which pairs of relations must be successively joined on their bounding boxes (spatial join) or alphanumerical attributes (relational join). This integrates in a common framework the handling of both types of data and illustrates the advantages of a constraint relational approach.

The filter step reduces drastically the amount of data subject to exact computation, and the algebra on bounding boxes provides a convenient tool to integrate the spatial access methods. Once the filter step is processed, the refinement step can be carried out by evaluating the algebraic expression q. This is investigated in the next sections.

It follows that the query processing techniques developed over existing spatial access methods can be reused in the constraint database context, while the query languages over constraint relations provide a new way to manipulate geometric data.

1.4 Query optimization

The task of the optimizer is to produce an efficient query evaluation plan (QEP) for each query. The design of an optimizer relies on the following components: *rewriting rules*, *control strategy* and *cost model*. First queries are usually represented as algebraic expressions which can be rewritten using *rewriting rules*. These rules define the search space for the QEP, as all the equivalent expressions for a given query. Since its size might not be bounded, a *control strategy* (based on heuristics) must be used to consider only those QEP which are likely to be the most efficient. Finally the choice is based on some meta-information (such as the size of the datasets or the existing access methods) and on *cost models* which yield the expected time-consumption of each candidate QEP.

The integration of spatial query evaluation techniques inside the query optimizer of a DBMS remains a real challenge. In the presence of new query processing techniques, a possible solution is to extend the optimizer with new rewriting rules, and to specify the strategy to follow in order to use these rules efficiently. This implies an algebraic description of the extended query language, a task likely to be tedious when the extension consists of operations over ADTs.

Contrary to spatial languages, constraint query languages are defined in the relational framework with simple algebras and their integration with existing rule-based optimizers is facilitated. Nevertheless, the decision problem associated to the equivalence of queries is undecidable. In the classical setting, it is well known that for conjunctive queries, the complexity of the containment, and the equivalence is NP-Complete. Only very few results have been proven yet for formalisms allowing arithmetic. It is known that in presence of order comparisons (e.g. $x \leqslant y$), the problem of query equivalence becomes Π_2^P-complete. More recently, it was shown that

Theorem 1 *The equivalence of constant-free conjunctive queries is decidable in double exponential time.*

The question of the decidability is still open for queries with constants. These results reveal the increase of the complexity of the query equivalence problem with arithmetic, thus indicating that the search of QEPs becomes more difficult than without arithmetic.

2 Impact of the data format

We now address the specific problem of efficiently representing spatial data as constraint relations. For the sake of practicality, we restrict the focus to a simple class of constraint databases, namely those which are definable by means of first-order formulas in the signature $\{\leqslant, +, 0, 1\}$ interpreted over some dense structure, such as the rationals or the reals. The restriction is justified by the similarity between the relations that can be represented by means of linear constraints, and the objects manipulated by multidimensional systems.

We first analyze the impact of the clustering of the constraints. The standard format for spatial data representation is based on lists of points describing frontiers of polygons. We study the mapping of this vector-based representation to the constraint one. If the latter requires the constraints to be in DNF, this means that a convexification should be performed. Other normal forms can be considered, with different properties.

We then study the evaluation of basic operations in terms of the dimension of the space. The complexity of several operations depends badly (exponentially) upon the dimension of the data. We present the concept of orthographic dimension, and various techniques that allow to manipulate data of some dimension d, at the cost of ℓ dimensional data ($\ell \leqslant$ d). In practice, since most of the existing algorithms are only feasible in the two dimension case, ℓ will be equal to 2.

2.1 Constraints clustering

Spatial data consisting of compact linear varieties (such as in computational geometry) can be easily represented as Boolean combinations of linear constraints. In dimension 2, convex polygons can be represented by a conjunction (intersection) of linear constraints (half-planes), and general polygons as a union of convex components. The choice of the format is crucial, because it determines the size of the constraint representation of the polygon and the algorithms that will be used to implement the basic algebraic operations. The size consists of the number of constraint tuples as well as the number of constraints that define each tuple.

Constraint databases have been initially designed with a representation of the relations by formulas in disjunctive normal form (DNF), where each disjunct is a constraint tuple. The symbolic representation is thus based on constraint relations consisting of finite sets of constraint tuples. The relation it represents is seen as a union of convex components. This normal form enforces a decomposition of the whole data into convex components. The number of convex components depends linearly on the number of *notches*[1] of the polygons. In particular, it is easy to see that the DNF format is not well suited for polygons with holes (see the right part of Figure 2). However, it offers a rather simple symbolic algebra.

Figure 2: DNF and CHNF representation of a polygon with hole

Other ways of clustering the constraints have been considered. As mentioned above, the increase of the database size resulting from the convexification can be lowered if holes are allowed in the data representation. A new logical representation allowing spatial objects with holes, which can themselves have holes generalizes the classical constraint approach. This data format offers strong advantages in practice while maintaining a low complexity of the basic operations.

A formula is in *convex with holes normal form* (CHNF) if it is either quantifier-free in DNF, or it is of the form $F - F'$, where F and F' are two formulas in CHNF of the same arity, and $-$ is the set difference (i.e. $\wedge\ \neg$). The left part of Figure 2 illustrates how a spatial object can be represented by a formula in CHNF. When only two convex polygons $q1$ and $q2$ represent the object in CHNF (it is seen as the *outside* polygon q_1 minus the *inner* one q_2), at least three convex polygons (in this case, the four convex polygons p_1, p_2, p_3 and p_4) are needed in the DNF format. We do not detail the algebra for relations in CHNF format and concentrate on the DNF format.

All relational algebraic operators can be expressed in a purely symbolic way (see Section 1). Symbolic operators take one or two formulas representing pointsets as input and yield a new formula which represents the result. The purely symbolic mode consists in computing a formula defining the output without making any efforts on the shape of the formula. However, it is often necessary to carry out a numerical evaluation in order to

[1]A notch is a vertex such that the internal angle is larger than π.

evaluate the pointsets represented by a formula. The following reasons motivate such an evaluation.

1. The size of intermediate results becomes very large (symbolic operators $\times$, $\cup$, $\cup$ or π_x do not decrease the number or the size of the formulas).

2. A conversion must be applied towards another representation (for graphical display purposes, etc.)

We introduce a new symbolic operator *simplify* that performs the arithmetic computation mentioned above. Applied to a constraint relation, it removes redundant constraints in constraint tuples, and checks for inconsistencies. The cost of each of these symbolic operator depends badly upon the format of the data, as explained in the following.

We first focus on the DNF data format. The costs of the symbolic operators in dimension 1, 2, 3 and d are given in the table below with respect to the following parameters: n is the total number of constraints in the relations, t is the number of tuples, and k the number of variables projected out. All complexities are given modulo a coefficient factor. The blow up in complexity comes from projection, set difference, and simplification.

Dimension $\rightarrow$ Operator $\downarrow$	1	2	3	d
$\times$	t^2	t^2	t^2	t^2
$\cup$	1	1	1	1
π_x	n	$n \log n$	n^4	$k2^{2^d}n^4$
$-$	$n \log n$	n^2	n^4	n^{d+1}
simplify	n	$n \log n$	$n \log n$	$(2^{2^d})n^2$

The symbolic algebra defined for manipulating data in CHNF format has a complexity comparable to the DNF case, with a cost-less difference, and a polynomial (quadratic) union.

As shown in the table above, the complexity of the operations depends badly upon the dimension. We next consider the impact of the dimension, and ways to master it.

2.2 Orthographic dimension

Although the data complexity of queries has been shown to be tractable, its dependance upon the dimension of the data seems to put a severe restriction on the ability of constraint database systems to handle efficiently high-dimensional relations. This section is devoted to techniques based on restrictions on the geometry of the spatial objects to overcome this problem. Intuitively, some problems present a decomposition such as space/time, or surface/altitude that can be used to lower the dimension upon which algebraic expressions operate. These properties are expressible as conditions on the formulas representing the pointsets. The approach presents an immediate advantage since the query evaluation process manipulates objects through their projections on lower dimensional subspaces, with a complexity which depends only linearly upon the dimension.

A quantifier-free formula with d free variables has *orthographic dimension (orthodim)* ℓ if there is a partition of the set of variables into (say k) *components*, $C_1, \cdots, C_k$, of at most ℓ variables each, such that each constraint in the formula can only involve variables from a single component. A relation of dim d has *orthographic dimension (orthodim)* bounded by ℓ if it can be represented by a quantifier-free formula of orthodim ℓ.

It is easy to see that a relation of orthodim bounded by ℓ is a finite collection of convex objects O, such that $O = \pi_{C_1} O \times \cdots \times \pi_{C_k} O$, where $\pi_{C_i} O$ is the projection of O on the ith component. $C_1, \cdots, C_k$ constitute an *orthographic decomposition.* Variables in different components are said *independent.* A convex d-dimensional object O with orthographic decomposition $\mathcal{D}$, is equal to the Cartesian product of all its projections on the components of $\mathcal{D}$. An example of such an object is represented in the figure on the right.

Note that the orthographic decomposition of a relation is not unique. Indeed, a relation can be defined by different formulas with distinct partitions of the variables.

In the contraint framework, relations are defined and manipulated by means of quantifier-free formulas. In general these formulas need not be of orthodim ℓ even if the intended relation is. Indeed a quantifier-free formula of orthodim ℓ can be equivalent to a quantifier-free formula that is not of orthodim ℓ. We identify the two following general problems of deciding whether there exists a representation of orthodim ℓ for a relation.

1. **Explicit orthographic dimension:**
   Given a quantifier-free formula in FO+LIN with d distinct variables, is there an equivalent formula with orthodim $\ell \leqslant d$.

2. **Implicit orthographic dimension:**
   Given a quantifier-free formula in FO+LIN with d distinct variables, is there a linear transformation of the system of coordinates, in which an equivalent formula with orthodim $\ell \leqslant d$ can be found.

Theorem 2 *Both problems of explicit and implicit orthographic dimension of a linear constraint relation defined by a linear formula can be solved in polynomial time.*

The relations of bounded orthographic dimension are very sensitive to some transformations. The class of relations of orthographic dimension ℓ is closed under set operations, projection, translation, symmetries (axial), but not under selections and rotations. The selection operator can relate by a single constraint a group of independent variables, thus possibly modifying the orthographic dimension of the input. For instance, if x and y are two independent variables in a relation R, then in $\sigma_{x \leqslant y}(R)$, x and y are in general dependent. Although rotations do not preserve the orthographic dimension, there are relations originally not of orthodim ℓ, which can be rotated adequately to become of orthodim ℓ, that is such that there exists a vector basis in which the relation has orthodim ℓ.

In the sequel we assume all data to be of dimension d with a fixed orthographic decomposition $\mathcal{D}$ and its corresponding $\ell \leqslant d$. A query *preserves the orthographic decomposition* $\mathcal{D}$ if for each input of orthographic decomposition $\mathcal{D}$, the ouput admits an orthographic decomposition which refines $\mathcal{D}$. We now explain how the evaluation process can take advantage of the orthographic dimension of an input, and exhibit queries on d-dimensional databases which can be solved using only manipulation of ℓ-dimensional pointsets.

It is easy to see, that the algebraic operators union, intersection, set difference, projection and Cartesian product preserve the orthographic decomposition of the input. Thus, they can be processed in parallel and independently over each component, and therefore only their implementation in the lower dimension is needed. The selection does not generally preserves the orthodim of the input. However, when a selection is used in a query that preserves the orthographic dimension, only an approximation of the selection is needed to compute the result of the query. We thus introduce a new selection operator which preserves the orthographic decomposition, and can be used in place of the classical selection.

In the sequel of the section, all queries preserve the orthographic decomposition $\mathcal{D}$ for some fixed orthographic decomposition $\mathcal{D}$. The technique is irrelevant for other queries.

The new symbolic selection, denoted by $\tilde{\sigma}_F$, computes, for each constraint tuple t, the projection $\pi_{C_i}(F \wedge t)$ for each component C_i and returns $\tilde{\sigma}_F(t) = \pi_{C_1}(F \wedge t) \times \ldots \times \pi_{C_n}(F \wedge t)$. Note that $\tilde{\sigma}_F(t)$ is not equal to $\sigma_F(t)$. It defines an approximation of $\sigma_F(t)$, such that the two coincide on each component.

To catch the intuition, consider a simple conjunctive preserving query $\mathcal{Q}(R) = \pi_z(\sigma_F(R))$, in the context of a relation of dimension 3 and orthodim 2, with decomposition $\mathcal{D} = (\{x, y\}, \{z\})$. We focus on the evaluation of q on a single tuple, O (see Figure 3.a), when the selection formula is respectively $F = x > z$ (Figure 3.b) and $F = x > z \wedge 2x + z < 2$ (Figure 3.c).

In Figure 3.b: the selection $\sigma_F(O)$, with $F = x > z$, yields a polytope O'. The exact value of O' is not relevant for the final output of the query which is $\pi_z(O')$: O' can be approximated by $\tilde{\sigma}_F(O)$ (in dashed lines on Fig. 3.b). Two comments are noteworthy. First, the result features a new constraint $z < \alpha$ where α is the highest coordinate, $B.z$, of O'. Note that the point B can be simply computed using F and the bounding-box of O. Second, a point in $\tilde{\sigma}_F(O)$, P for instance, might not belong to O', but its projections on $\{x, y\}$ and $\{z\}$ belong to $\pi_{x,y}(O')$ and $\pi_z(O')$ respectively.

In Figure 3.c, $F = x > z \wedge 2x + z < 2$ consists of two constraints. The same technique applies: we approximate the result of $\sigma_F(O)$ by the cross-product of its components. The new constraint in the final result is here $z < 2/3$, a constant value which results from the intersection of the half-planes $x > z$ and $2x + z < 2$. Note that it depends upon the query *and not upon the database* and can thus be computed only once.

Figure 3: Selection over a 3-d object of orthodim 2

ALG^ℓ is the set of algebraic expressions defined with standard union, $\cup$, and Cartesian product, $\times$, with projection, π^ℓ, negation, $-^\ell$, and simplification, $simplify^\ell$, restricted to inputs of dimension at most ℓ, and the above defined selection $\tilde{\sigma}_F$.

It is easy to see that FO $= \bigcup_\ell ALG^\ell$. On the other hand, if $\mathcal{D}$ (and thus ℓ) is fixed, it is conjectured that not all first-order queries on databases of orthographic decomposition $\mathcal{D}$ can be expressed in ALG^ℓ, even under the assumption that the query preserves the orthographic decomposition $\mathcal{D}$. Consider the relation $R(x, y, z)$ with orthographic decomposition $(\{x, y\}, \{z\})$, the relation $S(y, z)$, and the query defined as follows:

$$\mathcal{Q} = \pi_y(\underbrace{\pi_{y,z}(\sigma_{x<z}(R))}_{A} - S)$$

The sub-expression A is computed by $\pi^3_{y,z}(R \wedge x < z)$ which is in ALG^3 but it is not clear how to perform it within ALG^2. The latter example suggests that the hierarchy of ALG^ℓ is strict.

However, if we restrict our attention to Boolean combinations of conjunctive queries which preserve the orthographic decomposition $\mathcal{D}$ we can state the following fundamental result.

Theorem 3 *Let s be a database schema of orthographic decomposition $\mathcal{D}$ of orthodim ℓ. Let $\mathcal{Q}$ be a Boolean combination of conjunctive queries which preserve the orthographic decomposition $\mathcal{D}$ over s. Then $\mathcal{Q}$ can be easily rewritten in an equivalent query $\mathcal{Q}'$ in ALG^ℓ.*

All queries written in ALG^ℓ preserve the orthographic decomposition and the complexity of evaluating preserving queries is linear in d but exponential in ℓ only. Intuitively, they can be evaluated in parallel on each components, thus resulting in a complexity which is exponential only in the orthographic dimension and linear in d (maximum number of components). This is easy to see for $\cup$, $\times$, π^ℓ, $-^\ell$, and $simplify^\ell$, the main difficulty comes from $\tilde{\sigma}_F$. The symbolic evaluation of $\tilde{\sigma}_F$ is more complex than the one of σ_F but it is possible to prove that it remains polynomial in the data size, linear in d and exponential only in ℓ.

Corollary 4 *Let s be a database schema of orthographic decomposition $\mathcal{D}$ of orthodim ℓ. Let $\mathcal{Q}$ be a Boolean combination of conjunctive queries which preserve the orthographic decomposition $\mathcal{D}$ over s. Then $\mathcal{Q}$ can be evaluated in time polynomial in the data size, exponential in ℓ but linear in d.*

It is open whether the result extends to more general queries and, in general, it is undecidable if a query preserves a decomposition. However, Boolean combination of conjunctive queries, capture a large class useful queries. Thus, with a limited lost of representation power (i.e. the independence of variables), we can practically handle pointsets in arbitrary dimension while keeping a low evaluation cost and a system based on simple and robust algorithms. For applications which manipulate data in high dimension (such as spatio-temporal, or surfaces in 3d space), the orthodim concept applies naturally and offers a promising perspective. In practice, ℓ will be 2, because ALG^2 can be evaluated much more efficiently.

3 Query processing

Evaluation of queries over constraint data allows various modes of computation. First, the constraint based representation of data allows to perform a lazy evaluation of queries by maintaining (without solving them) the constraints defining the data. This will be called the *symbolic computation* mode. One advantage of this evaluation mode, is that it avoids, to some extent, the numerical precision problem. However, a conversion must be applied towards another simplified representation (for example for graphical display purposes) to actually evaluate the result of a query. The latter is performed by an *arithmetical computation* mode.

Geometric techniques can and should be used whenever possible to solve classical geometric problems. Indeed, efficient algorithms have been developped to solve specific geometric problems such as the computation of a convex hull. This raises the issue of identifying geometric problems that might be hidden in algebraic expressions. A geometric problem corresponds to an algebraic pattern in an expression and can be associated to a set of rewriting rules to identify its algebraic pattern. The use of the efficient algorithm to evaluate the identified pattern leads to a *geometric computation* mode.

When a query is known to be topological, the precise size of the data is no longer needed to evaluate it. For two dimensional data, it is possible to compute a finite annotation of the data, which is a planar graph, containing all the topological information contained in the database. This graph can be used to speed up the query evaluation process.

We therefore distinguish between the following fundamental modes of computation of spatial queries.

1. symbolic computation mode

2. arithmetic computation (constraint solving) mode

3. geometric computation mode

4. topological computation mode

The next section illustrates the reason why an efficient evaluation of a query requires the optimizer to produce a QEP based on the alternation of these computation modes.

In the last section, the query language FO+LIN is enriched with more primitives and nesting to express practical geometric queries. The nesting allows to deal with spatial objects as first class citizens, and the additional primitives are necessary to express basic spatial operations, such as convex hull. Algebraic pattern recognition is addressed with two geometric problems: the computation of a *convex hull*, and the *Voronoi decomposition*.

3.1 Alternation of computation modes

Spatial queries can be evaluated by completely different means. We illustrate in this section the need for alternation of computation modes with a query expressed by an algebraic expression. We analyze the cost of the operators in terms of the maximal number, n, of constraints per tuple, and the number, t, of tuples after computation of an operation, when alternating *symbolic evaluation* and *arithmetic evaluation*.

The use of constraints to represent data as well as the semantics of the algebra to express queries allow to carry on a lazy *symbolic evaluation* of the queries. The symbolic algebra defined for the DNF data format in Section 1.2 contains four operators that operate on the four imbricated levels of the formulas representing the relations. The formulas in DNF are under the following form:

$$\begin{array}{cccc} \bigvee_{i=1}^{k} & \bigwedge_{j=1}^{\ell_i} & \Sigma_k \alpha_{i,j}^k x_k + \alpha_0 & \Theta 0 \\ \uparrow & \uparrow & \uparrow & \uparrow \\ \cup & \times & \pi & - \end{array}$$

The symbolic operations operate as follows. Union operates on the (top level) disjunction, Cartesian product on the conjunction, set difference on the predicate of the constraints, and finally, projection on the parameters of the constraints themselves.

The *arithmetic evaluation* is performed by the oprator *simplify*. If R is a constraint relation, $simplify(R)$ does not affect the meaning of R (*conceptual representation*) but only changes its internal representation (*symbolic representation*). It eliminates redundancies and detects inconsistencies in each tuple of R separately. It returns a constraint relation such that all its constraint tuples are satisfiable. If R is inconsistent $simplify(R)$ will return an empty set. When an inconsistency is found in a constraint relation, the (top level) tuple having the inconsistent constraint relation as attribute can be considered as void.

The operation of simplification can be embedded anywhere in the query by the optimizer in order to reduce the size of the constraint relations involved. This might improve the time of query evaluation if the operation that follows has a bad complexity in term of the number of constraint of its inputs. After a selection, for instance, a lot of tuples are likely to be unsatisfiable and $simplify$ will drastically reduce the number of tuples involved. An arithmetic evaluation is also likely to be useful after operations such as projection and negation which output is of size at least quadratic of the size of its input with a lot of unsatisfiable tuples.

We consider the evolution of the size of the data in a simple example of a conjunctive query $\mathcal{Q}$, over some relations R (of arity 4) and P (of arity 2).

$$\mathcal{Q} \; : \; \pi_1(\sigma_{2=5\wedge 3=6}(R \times P))$$

The values in the array correspond to an evaluation with only final simplification, then with additional simplification after the selection, and finally, with a removal of overlapping tuples in the input for the ultimate case. The applications of operator $simplify$ corresponds to columns labeled by s.

evaluation		R	P	$\times$	σ	s	π_1	s
with final simplification	n	8	3	11	13		38	1
	t	3	4	12	12		12	2
and simplified selection	n	8	3	11	13	9	22	1
	t	3	4	12	12	2	2	2
and duplicate elimination	n	5	3	8	10	7	6	1
	t	6	4	24	24	5	5	2

This simple example clearly shows that both n and t drop if a simplification is performed after a selection. The decrease of n is even more glaring when an elimination of overlapping tuples is beforhand performed on the input. There is a trade-off between the number of tuples and the number of constraints per tuples in the case of the strong simplification (with non overlapping tuples). This trade-off can be in favor of the simplification in the case of a parallel evaluation.

Nevertheless, what really matters is the computation time, and again there is a trade-off between time and space, which can only be solved experimentally. The optimal solution lies between a lazy strategy with a unique final simplification, and a fully simplifying strategy with simplification after each operation. In practice $simplify$ will be performed at the end of each query evaluation.

3.2 Query pattern recognition

We conclude with some optimization principles which rely on the identification of query patterns or sub-patterns that are frequent in spatial queries. They correspond to geometric problems for which there are efficient algorithms. The sub-query corresponding to such a pattern should be rewritten in order to exploit efficient evaluation techniques.

In order to build some fundamental structures, we need to enrich the language with the primitives *axis* and *median*, defined as follows.

- $axis(\{p_1\},\{p_2\},\{p_3\})$, where the p_i's are points defines
  - undefined if $p_1 = p_2$
  - the line including p_1 and p_2 if p_3 is on the line, and otherwise

– the half plane delimited by the line p_1, p_2 which contains p_3.

- $median(\{p_1\}, \{p_2\}, \{p_3\})$, where the p_i's are points defines
  – undefined if $p_1 = p_2$
  – the median of p_1 and p_2 if p_3 is on this median, and otherwise
  – the half plane delimited by the median of p_1 and p_2 which contains p_3.

Given these new primitives, we illustrate the optimization principle in the sequel with some examples. We start with the convex hull of a finite set of points R, defined as a union of triangles constructed over a set of points by the following query.

$$unionest_1 MAP_{\lambda X.[Axis(X.1,X.2,X.3) \atop {\cap Axis(X.1,X.3,X.2) \atop \cap Axis(X.3,X.2,X.1)]}}(R \times R \times R)$$

The successive cross-products $R \times R \times R$ build all the possible triplets of points from R. The expression $Axis(\ldots) \cap Axis(\ldots) \cap Axis(\ldots)$ constructs then, for a given triplet denoted X, the associated triangle. Finally, *unionnest* groups these triangles.

The complexity of the naive evaluation of this query is in $O(n^3)$, where $n = |R|$, a rather bad complexity compared to the $O(n \log(n))$ complexity of efficient algorithms based on a divide-and-conquer strategy. If we are able to identify the potential of this expression for a divide-and-conquer computation, we should improve the query evaluation as well. This is what permits the identification of the generic pattern **ER1** defined as follows.

Let P be a finite set of points. Then the following equivalence rule holds.

$$\bigcup_{P^3} Triangle(\{x_1, x_2, x_3\}) \;=\; CH(P) \qquad \textbf{ER1}$$

where $CH(P)$ is the convex hull of P and $\bigcup$ is the geometric union of a set of triangles. Although unusual, the definition of the convex hull as the union of all triangles over k points can result from the declarativeness of the language.

The rewriting process relies on the set of rewriting rules depicted in Figure 4. It remains to show how we can identify **ER1** patterns when rewriting queries. Example 2 illustrates the technique with a the query computing the roads that come across the roman abbeys region in Provence.

Example 2 *Assuming a binary relation Abbeys containing the abbeys' names and their geometry (in a nested relation), and a binary relation Roads, the roman abbeys region corresponds to all the points that lies within three abbeys. The query can be expressed as follows:*

$$\pi_1 MAP_{\lambda X.[X.1, X.2 \cap unionnest_1(A)]}(Roads))$$

(a) Cascade of restructuring

$$MAP_{\lambda X.[X.1,...,X.i,alg_1(X.n_1,...X.n_j)oalg_2(X.m_1,...X.m_k)]}(R)$$
$$\equiv$$
$$MAP_{\lambda X.[X.1,..,X.i,X.i+1oalg_2(X.m_1,..,X.m_k)]}(MAP_{\lambda X.[X.1,..,X.i,alg_1(X.n_1,..,X.n_j),X.m_1,..,X.m_k]}(R))$$

where alg_1 and alg_2 are any valid algebraic expressions yielding set of points, and "o" is either "," or a binary set operator.

(b) Commuting MAP and $\times$

$$MAP_{\lambda X.[alg_1(X.i_1,...,X.i_n),alg_2(X.j_1,...,X.j_m)]}(R \times S)$$
$$\equiv$$
$$MAP_{\lambda X.[alg_1(X.i_1,...,X.i_n)]}(R) \times MAP_{\lambda X.[alg_2(X.j_1,...,X.j_m)]}(S)$$

for any expressions alg_1 and alg_2, when $X.i_{p,p\in[1,n]} \in sort(R)$ and $X.j_{q,q\in[1,m]} \in sort(S)$.

(c) Pushing *internest* or *unionnest* down $\times$

$$OP_i(R_1 \times R_2) \equiv R_1 \times OP_{i-k}(R_2)$$

where $OP \in \{internest, unionnest\}$, $arity(R_1) = k$, $i > k$ and there are no duplicates in R_1.

(d) Commutativity of MAP and *internest* or *unnest*

$$MAP_{\lambda X.[X.1,...,X.i-1,OP_1(MAP_{\lambda Y.[alg(\{X.i\},\{Y.j\})]}(R_2))]}(R_1)$$
$$\equiv$$
$$OP_i(MAP_{\lambda X.[X.1,...,X.i-1,alg(\{X.i\},\{X.i+j\})]}(R_1 \times R_2))$$

where $OP \in \{internest, unionnest\}$ and alg is any expression over $att(R_1) \cup att(R_2)$ which yields an attribute of type $\{Q^k\}$

Figure 4: Subset of possible rewriting rules for nested algebra

where A is

$$MAP_{\lambda Y.[Triangle(Y.2,Y.4,Y.6)]}(Abbeys \times Abbeys \times Abbeys)$$

The naive evaluation would result in a $O(n \times m^3)$ complexity, where $n = |Roads|$ and $m = |Abbeys|$. Let's see how rewriting can improve this.

¿From rule (a) in figure 4 one obtains:

$$\pi_1 MAP_{\lambda X[X.1,X.2\cap X.3]}(B)$$

where B is :

$$MAP_{\lambda Y[Y.1,Y.2,unionnest_1(MAP_{\lambda Z.[Triangle(Z.2,Z.4,Z.6)]}(Abbeys^3))]}(Roads)$$

Rule (d) holds for B which becomes:

$$unionnest_3(MAP_{\lambda Y.[Y.1,Y.2,Triangle(X.4,X.6,X.8)]}(Roads \times Abbeys^3))$$

Finally, by rules (b) and (c) one obtains:

$$MAP_{\lambda Y.[Y.1,Y.2]}(Roads) \times C$$

where C is :

$$unionnest_1(MAP_{\lambda X.[Triangle(X.2,X.4,X.6)]}(Abbeys \times Abbeys \times Abbeys))$$

The sub-expression C in the above query matches the convex hull pattern defined in rule **ER1**. *We are now able to compute separately the convex hull in $O(n \log n)$, and the intersection with roads is linear in $|Roads|$.*

We now consider the Voronoi decomposition, which corresponds to another class of useful problems. Let P be a finite set of points. The following equivalences hold.

$$\bigcap_{Y \in P} median(X,Y,X) = VP(X,P) \qquad \textbf{ER2a}$$

and

$$\{[X, \bigcap_{Y \in P} median(X,Y,X)] | X \in P\} = VD(P) \qquad \textbf{ER2b}$$

Rule **ER2a** states that the intersection of all half planes defined by all medians of segments between a given point X and each point Y of a set P, defines the Voronoi polygon of X with respect to P, $VP(X,P)$. The Voronoi diagram, $VD(P)$, of all points X in P is defined by rule **ER2b**.

The application of the previous rules is illustrated in Example3 with the optimization of a nearest neighbor query.

Example 3 *Assume two relations $Waterpoints$ and $Cities$ that both contain a name and a nested relation corresponding to the geometry and the query $\mathcal{Q}$ that returns the nearest waterpoint from each city. $\mathcal{Q}$ can be expressed as follows:*

$$\pi_{1,2} MAP_{\lambda X.[X.1,X.3,X.4,A]}(Cities \times Waterpoint)$$

where A is :

$$X.2 \cap internest_1(MAP_{\lambda Y.[median(X.4,Y.2,X.4)]}(Waterpoint))$$

Using various rewriting rules, this query can be shown to be equivalent to

$$Cities \times internest_3(MAP_{\lambda Z[Z.1,Z.2,median(Z.2,Z.4,Z.2)]}(Waterpoint^2))$$

*The pattern $internest_i(MAP_{\lambda X.[X.1,\ldots,median(X.j,X.k)]}(R \times R))$ defines the Voronoi diagram of the point-set R (***ER2***).*

In summary, identifying geometric patterns in algebraic expressions allows to use a general and non-specialized query language yet preserving the ability to trigger efficient algorithms whenever some fundamental pattern occurrences are detected. Together with the ability to alternate symbolic and arithmetical computation, this yields a rich and promising framework for establishing optimization principles.

Bibliographic notes

Query processing and optimization in spatial databases has been addressed in numerous papers, see for instance [Ore86, OM88, OSDD89, Ooi90]. Main principles were proposed in [Ore86], and reviewed in details in [BKSS94]. References for spatial access methods can be found in the recent survey [GG98]. Research on spatial joins processing is a very active field: some recent proposals are [BKS93, LR94, LR96, PD96, KS97]. A description of some extensible optimizers can be found in [Fre87, GD87]. The integration of spatial query optimization in an extensible optimizer is detailed in [BG92] for the GRAL system [Güt88]. Cost models have been recently proposed in [TS96, TSS98].

The optimization of queries over constraint databases has been studied only very recently. In [GL95], a worst case analysis of the evolution of the size of the representation is performed in the case of dense linear constraint queries.

The data complexity of linear constraint queries has first been investigated in [KKR95], where it was shown more generally that polynomial constraint queries had NC data complexity. In the linear case, the complexity is actually of NC^1 [GS97b]. It has also been shown in [GST95] that the data complexity of linear constraint queries restricted to k-bounded classes of linear constraint inputs is in AC^0 (a class of linear constraint relations is k-bounded if each individual linear constraint appearing in a tuple of a relation contains at most k occurrences of the addition symbol, and only integer parameters).

Most of the complexity result of the operators can be found in the book [GO97] (chapter 19). The orthographic dimension has been introduced in [**?**]. Its application in the case of spatio-temporal data has been shown in [GRS98a]. The concept of variable independence was originally introduced in [CGK96]. The rewriting of DEDALE queries was first introduced in [GRS98b].

In a more general setting, [GS96] studies the use of approximate methods to evaluate queries over polynomial constraint databases.

The containment and equivalence of constraint queries was studied in particular [GS97a, IS97]. Theorem 1 was shown in [IS97].

Nested algebra were introduced in [BBC97, GRS98b]. A more complete set of rewriting rules is presented in the long version of [GRS98b].

References

[BBC97] A. Belussi, E. Bertino, and B. Catania. Manipulating spatial data in constraint databases. In M. J. Egenhofer and J. R. Herring, editors, *Intl. Conf. on Advances in Spatial Databases (SSD'97)*, pages 115–141. Springer Verlag, LNCS 1262, 1997.

[BG92] L. Becker and R.H. Güeting. Rule-Based Optimization and Query Processing in an Extensible Geometric Database System. *ACM Transactions on Database Systems*, 17(2):247–303, October 1992.

[BKS93] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient Processing of Spatial Joins Using R-Trees. In *SIGMOD*, 1993.

[BKSS94] T. Brinkhoff, H.-P. Kriegel, R. Schneider, and B. Seeger. Multi-Step processing of Spatial Joins. In *SIGMOD*, pages 197–208, 1994.

[CGK96] J. Chomicki, D. Goldin, and G. Kuper. Variable Independence and Aggregation Closure. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 40–48, Montréal, Canada, June 1996.

[Fre87] J. C. Freytag. A rule-based view of query optimization. In *SIGMOD*, pages 173–180, San Francisco, 1987.

[GD87] G. Graefe and D. DeWitt. The EXODUS optimizer generator. In *SIGMOD*, pages 160–172, San Francisco, 1987.

[GG98] V. Gaede and O. Guenther. Multidimensional Access Methods. *ACM Computing Surveys*, 1998. Also available at http://www.icsi.berkeley.edu/ oliverg/survey.ps.Z.

[GL95] S. Grumbach and Z. Lacroix. Computing queries on linear constraint databases. In *Fifth Int. Workshop on Database Programming Languages (DBPL'95)*, Electronic Workshops in Computing. Springer, 1995.

[GO97] Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.

[GRS98a] S. Grumbach, P. Rigaux, and L. Segoufin. Spatio-Temporal Data Handling with Constraints. In *ACM-GIS conference*, 1998.

[GRS98b] S. Grumbach, P. Rigaux, and L. Segoufin. The dedale System for Complex Spatial Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Estes Park, Colorado, USA, August 1997 1998.

[GRS99] S. Grumbach, P. Rigaux, and L. Segoufin. On the Orthographic Dimension of Constraint Databases. In *ICDT*, 1999.

[GS96] S. Grumbach and J. Su. Towards practical constraint databases. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 28–39, New York, 1996. ACM Press.

[GS97a] S. Grumbach and J. Su. Finitely representable databases. *Journal of Computer and System Sciences*, 55(2):273–298, 1997.

[GS97b] S. Grumbach and J. Su. Queries with arithmetical constraints. *Theoretical Computer Science*, 173(1):151–181, 1997.

[GST95] S. Grumbach, J. Su, and C. Tollu. Linear constraint query languages: expressive power and complexity. In D. Leivant, editor, *Logic and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 426–446. Springer, 1995.

[Güt88] R. Güting. Geo-relational algebra: A model and query language for geometric database systems. In *Proceedings of Extending Data Base Technology*, pages 506–527, 1988.

[IS97] O. Ibarra and J. Su. On the containment and equivalence of database queries with linear constraints. In *ACM Symp. on Principles of Database Systems*, 1997.

[KKR95] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51:26–52, 1995.

[KS97] N. Koudas and K. Sevcik. Size separation spatial join. In *SIGMOD*, pages 324–335, 1997.

[LR94] M.-L. Lo and C.V. Ravishankar. Spatial Joins Using Seeded-Trees. In *SIGMOD*, 1994.

[LR96] M.-L. Lo and C.V. Ravishankar. Spatial Hash-Joins. In *SIGMOD*, pages 247–258, 1996.

[OM88] J.A. Orenstein and F.A. Manola. Probe spatial data modeling and query processing in an image database application. *IEEE Trans. on Software Engineering*, 14(5):611–629, 1988.

[Ooi90] B.C. Ooi. *Efficient Query Processing in a Geographic Information System.* LNCS No. 471. Springer-Verlag, 1990.

[Ore86] J. A. Orenstein. Spatial Query Processing in an Object-Oriented Database System. In *SIGMOD*, pages 326–336, 1986.

[OSDD89] B.C. Ooi, R. Sack-Davis, and K.J. Mc Donell. Extending a DBMS for Geographic Applications. In *Proc. Intl. Conf. on Data Engineering*, pages 590–597, 1989.

[PD96] J.M. Patel and D. J. DeWitt. Partition Based Spatial-Merge Join. In *ACM-SIGMOD Intl. Conf. on Management of Data*, pages 259–270, 1996.

[TS96] Y. Theodoridis and T. Sellis. A Model for the Prediction of Rtree Performance. In *PODS*, 1996.

[TSS98] Y. Theodoridis, E. Stephanakis, and T. Sellis. Cost Models for Join Queries in Spatial Databases. In *ICDE*, 1998.