The DEDALE System for Complex Spatial Queries*

Stéphane Grumbach
Università di Roma Tre
and INRIA

Philippe Rigaux
CNAM

Luc Segoufin
INRIA

technical report

Abstract

This paper presents DEDALE, a spatial database system intended to overcome some limitations of current systems by providing an abstract and non-specialized data model and query language for the representation and manipulation of spatial objects. DEDALE relies on a logical model based on linear constraints, which generalizes the constraint database model of [KKR90]. While in the classical constraint model, spatial data is always decomposed into its convex components, in DEDALE holes are allowed to fit the need of practical applications. The logical representation of spatial data although slightly more costly in memory, has the advantage of simplifying the algorithms. DEDALE relies on nested relations, in which all sorts of data (thematic, spatial, etc.) are stored in a uniform fashion. This new data model supports declarative query languages, which allow an intuitive and efficient manipulation of spatial objects. Their formal foundation constitutes a basis for practical query optimization. We describe several evaluation rules tailored for geometric data and give the specification of an optimizer module for spatial queries. Except for the latter module, the system has been fully implemented upon the O_2 DBMS, thus proving the effectiveness of a constraint-based approach for the design of spatial database systems.

1 Introduction

The introduction of spatial information in database systems raises important and specific issues. Apart from performance considerations due to the large volume of spatial data, the main challenge is the design of data models general and powerful enough to handle both conventional data and spatial objects. The DEDALE system offers a sound data model which allows a uniform representation of all sorts of data, and supports declarative query languages well-suited for complex spatial queries. A powerful SQL like query language has been designed. Its advantages over most spatial query languages is to be extensible (e.g. no limitation in the dimension), simple (non expert users allowed), and independent from the physical level and the algorithms (potential for query optimization). With respect to standard constraint database approaches, the DEDALE framework offers better spatial data modeling, thanks to the nested relations and the explicit representation of holes.

General purpose database management systems (DBMS), e.g. relational DBMSs, are not appropriate for storing and manipulating spatial data, in particular because of the complex structure

*Work supported in part by TMR project Chorochronos.

[0]Contact author: Luc Segoufin Tel +33 1 39 63 56 74, Fax +33 1 39 63 56 79
[0]INRIA, Rocquencourt BP 105, F-78153 Le Chesnay, France — {Stephane.Grumbach,Luc.Segoufin}@inria.fr
[0]Cedric/CNAM, 292 rue St Martin, F-75141 Paris Cedex 03, France — rigaux@cnam.fr

of geometric information and to the intricate topological relationships among sets of spatially related objects. Moreover, the costly operations involved in geometric object management seems at first glance to prevent the logical-level approach based on simple data structures (e.g. relations) manipulated through a limited set of simple operations (e.g. relational algebra). The main trend has therefore been to introduce, as additional logical components of the database model, carefully designed geometric structures associated with computational geometry algorithms in order to provide an efficient evaluation of spatial operations [Tom90].

Partly because of their specific implementation of spatial object management, most of the existing Geographic Information Systems (GIS) rely on a strict distinction between alpha-numerical data and geometric data. For instance, Arc/Info [Mor89] extends an efficient module (ARC) for manipulating geometric and topologic information with a relational module (INFO) for manipulating alphanumeric information associated with geometric objects. Many research models and prototypes favor ad hoc algebras and extensions of conventional database models with abstract spatial data types encapsulating geometric structures and operations (see for example [RFS88, OM88, SV89, Güt89, GS95]). Even commercial DBMS such as Oracle [Her96] and Illustra [Ube94] provide in their latest version a separate module devoted to spatial data. The common drawback of these systems is the lack of uniformity for the representation of data.

The constraint data model, first introduced by Kanellakis, Kuper and Revesz [KKR90] offered a promising paradigm for the representation of all sorts of data in a unified framework. Linear constraints over rational numbers have been shown to fit the need of spatial data in the vector mode [KPV95, GK97]. Spatial objects, seen as infinite sets of points, could be dealt with as first class citizens with an explicit representation. This contrasts with the representation of spatial objects by their boundary in vector mode for instance, which lead to cumbersome data models with ad hoc operations, and no standard emerging. The fundamental difference between the two approaches relies in the explicit definition of the spatial objects in the data model (e.g. a polygon is explicitly the infinite set of points it contains versus the implicit definition by the sequence of border points). It is now possible to manipulate spatial objects through standard set operations (e.g. those of relational algebra). The user disposes from say an intersection primitive, that the system will evaluate differently depending upon the input.

The basic idea of the constraint model is to represent spatial objects as infinite collections of points satisfying first-order formulae. For instance, a convex polygon, which is the intersection of a set of half-planes, is defined by the conjunction of the inequalities defining each half-plane. A non convex polygon by the union (logical disjunction) of a set of convex polygons. In the initial framework of [KKR90] and subsequent publications in constraint databases, the formulae are assumed to be in disjunctive normal form (DNF), thus leading to a decomposition of spatial objects into convex components. Although this does not restrict the kind of spatial objects which can be stored in the database, it forces to decompose them into convex components while loading the database. Such a decomposition increases the size of the database [GRSS97], but simplifies the subsequent manipulation of the spatial objects during query evaluation. It is still an open problem to know if the trade-off is in favor of the convexification of the data.

Nevertheless, in practice, the increase of the database size resulting from the convexification can be lowered if holes are allowed in the data representation. We therefore introduce a new logical representation (CHNF) which generalizes the classical constraint approach, by allowing spatial objects with holes, which can themselves have holes. This new data model, offers strong advantages in practice while maintaining a low complexity of the basic operations.

Finally, the data model of DEDALE provides non-first normal form relations with one level of nesting. Alphanumeric information can be easily represented in unnested attributes while one level of set nesting can be used for spatial data. Such a model has also been proposed in [BBC97].

The solid logical foundations of the data model allows the design of declarative query languages, which are not application specific, and do not require expert users unlike most of the query languages provided in spatial information systems. We follow the trend initiated with constraint databases [PVV94, GST94, KG94], and develop a first-order (FO) based algebraic query language with additional functionalities related both to the particular data structures of the model and to some fundamental spatial operators. Indeed, it is well known that while FO languages with constraints enjoy a low data complexity [GS97], they suffer from a restricted expressive power: For instance, neither topological connectivity nor metric operations such as distance can be expressed. Therefore we introduce in the language some metric and topological operators which significantly increase its expressive power, yet preserving the fundamental feature that both inputs and outputs of algebraic operators are linearly representable. The result is a powerful algebra that covers most of the classical operations required in spatial databases. In addition, fundamental geometric structures and operations such as convex hull and Voronoi diagrams can be expressed in a natural way, as well as some intricate queries addressing topological relationships.

A simple SQL like query language is also proposed. It allows the user to write complex spatial queries, in a declarative mode, without thinking of the algorithmic aspects of the queries. We illustrate the power of the language through a large list of queries from GIS, spatio-temporal applications, and robotics related problems. Queries like "nearest neighbor" or "visibility" (see section 4) which are usually difficult to express in other languages can easily be expressed in DEDALE.

One of the most important characteristics of the DEDALE algebra is the fact that it can be optimized. The optimization of query languages in current spatial database systems has been insufficiently addressed. Indeed, in most cases, either there is no high level query language (such as in Arc/Info) and the user has to write the proper sequence of language primitives to be efficient, or the system architecture is based on the extension of a conventional DBMS with geometric operations, which renders the DBMS optimization module obsolete in the presence of spatial criteria, leading to drastically inefficient execution plans. We propose new evaluation techniques which differ from the classical ones and take into account the geometric nature of the data. In particular, we show how to recognize patterns in queries corresponding to specific problems for which there are very efficient algorithms, such as the convex hull. Spatial databases are usually very large and therefore, evaluating queries is time consuming. To improve the efficiency, parallel evaluation of spatial database queries was studied in [Ja97]. Because the DEDALE query language is based on first-order, it potentially lends itself to a parallel implementation.

DEDALE has been implemented on top of the O_2 object DBMS [BDK92]. In the current prototype, query evaluation partly relies on the standard OQL [BCD89]. Constraint operations that manipulate sets of points have been fully implemented, together with multidimensional access paths: this yields a set of physical operations and data structures which has already been validated. The implementation of the operations of relational algebra over spatial relations has been presented in [GRSS97]. A more advanced DEDALE query evaluator following the optimization rules and principles presented in the sequel is under implementation.

Two geographic applications run on DEDALE. The first one is based on a dataset produced by the French Institute for Geography, IGN, and the second one from the SEQUOIA project [SFGM93]. The queries presented in section 4 have been successfully evaluated on both applications. The system DEDALE demonstrates the possibility to build efficient spatial database systems that, unlike the classical approach, respect the fundamental distinction between logical and physical levels.

The paper is organized as follows. In the following section, we present the logical framework for the representation of spatial data with holes. Section 3 introduces the DEDALE nested data model, together with the algebra. The user query language, along with some queries which illustrate the power of the language in the context of various applications, is described in Section 4. The architecture of DEDALE is presented in Section 5, with guidelines for the design of specific optimization rules for operations on spatial data.

2 Logical manipulation of spatial data

In this section, we present a logical framework for the representation of spatial data, modeled as infinite relations which admit a finite representation in terms of first-order formulae, and define relational algebra operators on spatial relations.

We consider a very general framework for spatial data, in which data is modeled as infinite sets in the rational space. For example, a polygon in the plane is seen as the infinite set of points of $\mathbb{Q}^2$ inside its frontier, and a 3-dimensional pyramid is seen as the infinite set of points of $\mathbb{Q}^3$ inside its facets.

At this *abstract level*, spatial data consists of infinite relations over the universe of the rational numbers $\mathbb{Q}$. These infinite relations can only be described and manipulated through a finite representation. Following the trends of constraint databases [KKR90], we use first-order logic to represent the relations of interest, and define a *symbolic level* of representation.

We consider linear constraints in the first-order language $\mathcal{L} = \{\leqslant, +\} \cup \mathbb{Q}$ over the structure $\mathcal{Q} = \langle \mathbb{Q}, \leqslant, +, (q)_{q \in \mathbb{Q}} \rangle$ of the linearly ordered set of the rational numbers with rational constants and addition. Constraints are linear equations and inequalities of the form: $\sum_{i=1}^{p} a_i\, x_i \Theta a_0$, where Θ is a predicate among $=$ or $\leqslant$, the x_i's denote variables and the a_i's are integer constants. Note that rational constants can always be avoided in linear equations and inequalities. The multiplication symbol is used as an abbreviation, $a_i x_i$ stands for $x_i + \cdots + x_i$ (a_i times).

Let $\sigma = \{R_1, \ldots, R_n\}$ be a database schema such that $\mathcal{L} \cap \sigma = \emptyset$, where $R_1, \ldots, R_n$ are relation symbols. We distinguish between *logical predicates* (e.g., $=, \leqslant$) in $\mathcal{L}$ and *relations* in σ. Linear constraint relations are defined as follows.

Definition 1 Let $S \subseteq \mathbb{Q}^k$ be a k-ary relation. The relation S is a linear constraint relation if there exists a formula $\varphi(x_1, \ldots, x_k)$ in $\mathcal{L}$ with k distinct free variables $x_1, \ldots, x_k$ (called a *representation* of S) such that:

$$\mathcal{Q} \models \forall x_1 \cdots x_k (S(x_1, \ldots, x_k) \leftrightarrow \varphi(x_1, \ldots, x_k))$$

The class of linear constraint relations constitutes a rather drastic restriction over the class of all infinite relations over $\mathbb{Q}^k$. This restriction is necessary to ensure reasonable query complexity and is sufficient to encompass spatial data in computational geometry, GIS, etc. as it has already been widely demonstrated in the literature.

More drastic restrictions have been traditionally imposed in constraint databases [KKR90]. The first one is to represent relations by quantifier free formulae. This restriction is harmless, since for each formula, there is an equivalent quantifier free formula, which moreover can be constructed effectively. Therefore each spatial object can be represented by a quantifier free formula. The rationale for the quantifier free representation is the resulting reasonable query complexity. The second restriction enforces the formula to be in disjunctive normal form (DNF). This restriction is harmless as well, since each formula can be put in DNF, and again the motivation is complexity.

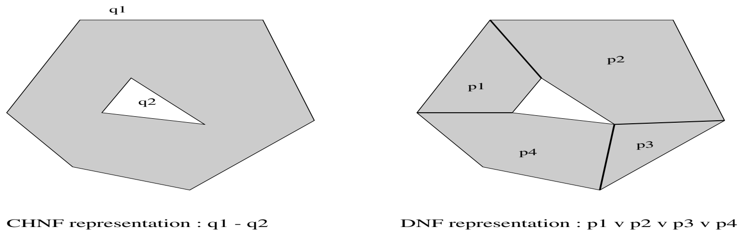

Figure 1: DNF and CHNF representation of a polygon with hole

A formula φ in DNF :

$$\varphi \equiv \bigvee_{i=1}^{k} \bigwedge_{j=1}^{\ell_i} \varphi_{i,j}$$

is seen as a *generalized relation*, that is a finite set of *generalized tuples*, which are conjunctions of atomic formulae $\varphi_{i,j}$

$$\left\{ t_i \;\middle|\; 1 \leqslant i \leqslant k,\ t_i = \bigwedge_{j=1}^{\ell_i} \varphi_{i,j} \right\}$$

The representation in terms of DNF formula enforces a specific view of spatial objects. Indeed, each relation is the union (disjunction) of a finite set of convex objects (generalized tuples). In terms of 2-dimensional applications, this implies for example that polygons must be decomposed into convex component polygons (convexified). We call the previous restriction the *convex normal form.*

Again note that the convexification doesn't restrict the class of spatial objects that can be represented, but imposes a transformation of the objects prior to their manipulation, namely decomposition into convex components.

We generalize this restriction, motivated both by its logical simplicity and its algorithmic efficiency, to a representation of objects with "holes", a situation frequent in many spatial applications.

Definition 2 A formula is in the *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 formulae in CHNF of the same arity, and $-$ is the set difference (i.e. $\wedge\neg$).

Figure 1 shows how a spatial object can be represented by a formula either in DNF or in CHNF. In DNF it is seen as the union of the four polygons p_1, p_2, p_3, p_4 while in CHNF it is seen as the *outside* polygon p_1 minus the inner one p_2.

A *(database) instance (of σ)* is a mapping which associates to each k-ary relation symbol R in σ a linear constraint relation of arity k.

We have exhibited above two views (normal forms) of the relations at the symbolic level. We now consider relational algebraic operators applied to linear constraint relations, and their variants depending upon the normal forms. We consider the algebra introduced by Codd for finite relations [Cod70], in the case of infinite relations, i.e. operators apply to any relation, whether it is finite or infinite. We refer to this algebra at the abstract level over infinite relations as the *abstract algebra.*

It was shown in [GST94] that the abstract algebra is equivalent to first-order logic over the class of linear constraint databases. The proof is quite similar to that of the equivalence of the classical relational algebra and calculus over finite structures. The combination of selection and Cartesian product can yield complicated forms of selections.

For each of the three symbolic representations (quantifier free formulae with no normal form assumption, DNF and CHNF formulae), we consider a *symbolic algebra*, with operators applying to formulae representing possibly infinite sets, and which return formulae of the same symbolic representation. The effect of the operators on formulae emulates their effect on infinite relations.

We first consider the quantifier-free representation with no normal form assumption. The symbolic algebra contains the following operations: intersection, $\cap$, projection, π, union, $\cup$, set difference, $-$, and rename, ρ. Let F_1 and F_2 be two first-order formulae.

1. $F_1 \cap F_2$ is defined by $F_1 \wedge F_2$.
2. $\pi_{\overline{x}}\ F_1$ is defined by eliminating the quantifiers in the formula $\exists \overline{y} F_1$, where $\overline{y}$ are the free variables of F_1 that are not in $\overline{x}$.
3. $F_1 \cup F_2$ is defined by $F_1 \vee F_2$.
4. $F_1 - F_2$ is defined by $F_1 \wedge \neg F_2$.
5. $\rho_{[x,x']}\ F_1$ is defined by $F_1[x, x']$ where variable x is replaced by variable x'.

With those operators it is possible to get classical operations on two relations R_1 and R_2, respectively represented by formulas F_1 and F_2.

1. $R_1 \times R_2$ is defined by $F_1 \cap F_2$ (F_1 and F_2 have distinct variables).
2. $\pi_{\overline{x}}\ R_1$ is defined by $\pi_{\overline{x}}\ F_1$.
3. $R_1 \cup R_2$ is defined by $F_1 \cup F_2$ (F_1 and F_2 have the same variables).
4. $R_1 - R_2$ is defined by $F_1 - F_2$ (F_1 and F_2 have the same variables).
5. $R_1 \cap R_2$ is defined by $F_1 \cap F_2$ (F_1 and F_2 have the same variables).
6. $\sigma_F R_1$ is defined by $F_1 \cap F$.
7. $\rho_{[A,A']}\ R_1$ is defined by $\rho_{[x,x']}\ F_1$, where x (resp. x') is the variable corresponding to attribute A (resp. A').

The complexity of the above operations is the following. The operations $\times$, $\cap$, $\cup$, σ and $-$ are calculated in constant time on the size of the formulae, and the operation $\pi_{\overline{x}}\ F$ is calculated using algorithm given by [KY85] which is roughly $O(n^{2^k})$ where n is the size of F and k the number of variables to be eliminated.

Two additional operations are fundamental in the symbolic algebra. First, the Boolean *emptyness test*, $\emptyset^?$, operation which returns the empty tuple, $\{\ [\ ]\ \}$ if the formula is not satisfiable, and returns the emptyset, $\{\ \}$, otherwise. More important, *normalization*, ν recursively transforms an input quantifier-free formula into a simpler one by (i) removing redundant terms in conjunctions (i.e. terms implied by the rest of the conjunction), and replacing non satisfiable conjunctions by *False*, and (ii) removing *False* disjuncts in disjunctions. It can be verified that if F is any quantifier-free formula, $\nu(F) = False$ iff $\emptyset^?(F) = \{\ \}$.

For a quantifier-free formula, the complexity of the emptyness test is in $O(n^{2^k})$ where k is the arity of the relation and n the size of the formula defining it.

We focus now on the algebra for relations in normal form. As studied below, the normal forms motivated by 2-dimensional spatial applications allow to lower the complexity of some among the previous tests and operations. Depending on which operation is favored, one of the above normal forms might be chosen as a representation.

The operations of the symbolic algebra admit simpler definitions in the case of relations in normal form. We first consider relations in convex normal form (in DNF). The algebra operations are performed on sets of generalized tuples, i.e. on quantifier-free formulas in *disjunctive normal form* (union of convex objects).

As above five operations: $\cap$, π, $\cup$, $-$, ρ, are defined. Let F_1 and F_2 be two DNF formulas. F_i can be considered as a set e_i of generalized tuples.

1. $F_1 \cap F_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in e_2\}$.

2. $\pi_{\overline{x}}\, F_1 = \{\pi_{\overline{x}}\, t \mid t \in e_1\}$,
   where
   $$\pi_{\overline{x}}\, t = \bigwedge_{1 \leqslant k \leqslant K, 1 \leqslant \ell \leqslant L} b^k\overline{x} - b_0^k \leqslant a_0^\ell - a^\ell\overline{x} \;\wedge\; \bigwedge_{1 \leqslant i \leqslant I} c^i\overline{x} \leqslant c_0^i.$$
   is given by the Fourier-Motzkin Elimination method [Sch86] from a tuple t defining a polyhedron $P(\overline{x}, y) \subseteq \mathbb{Q}^{n+1}$ described by the inequalities (once the coefficients of y have been isolated):
   $$\begin{cases} a^\ell\overline{x} + y \leqslant a_0^\ell & \text{for } \ell = 1, \ldots, L \\ b^k\overline{x} - y \leqslant b_0^k & \text{for } k = 1, \ldots, K \\ c^i\overline{x} \leqslant c_0^i & \text{for } i = 1, \ldots, I \end{cases}$$
   where $\overline{x}$ ranges over $\mathbb{Q}^n$, and y over $\mathbb{Q}$.

3. $F_1 \cup F_2 = e_1 \cup e_2$.

4. $F_1 - F_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in (e_2)^c\}$,
   where e^c is the set of tuples or disjuncts of a DNF formula equivalent to $\neg e$.

The previous symbolic operators are sufficient to emulate the abstract relational algebra, in the same way as above for quantifier-free formulae.

The complexity of the operations on DNF formulae is simpler than in the general case. $\cup$ is calculated in constant time on the size of the formulas. $\pi_{\overline{x}}\, F$ is calculated using the Fourier-Motzkin algorithm which is in n^{2^l} where l is the number of variables eliminated and n the number of constraints in F (the projection is quadratic when eliminating one variable, but the size of the output is also quadratic in the input size). $\cap$ is quadratic. $-$ is in time n^2 in dimension 2, otherwise it is exponential in the dimension. The complexity of the emptyness test is $O(n \log n)$ in dimension 2. Normalization is defined as eliminating empty tuples and redundancies inside each tuple. It can be done in time $O(n \log n)$ in dimension 2.

Finally we consider the operations of the symbolic algebra for relations in the convex with holes normal form (CHNF). The algebra contains the following operators, $\cap$, $\cup$, $\times$, σ, π, $-$, and ρ. Let $F_1 = \alpha - \alpha'$ and $F_2 = \beta - \beta'$ be two formulas in CHNF, where α, α', β and β' are CHNF.

1. $F_1 \cap F_2$ is defined recursively by $(\alpha \cap \beta) - (\alpha' \cup \beta')$.

2. $F_1 \times F_2$ is defined recursively by $(\alpha \times \beta) - ((\alpha' \times \beta) \cup (\beta' \times \alpha))$.

	Symbolic Algebras					
domain $\rightarrow$	QF formulae		DNF formulae		CHNF formulae	
operator $\downarrow$	effect	comp.	effect	comp.	effect	comp.
$\cap$	$F_1 \wedge F_2$	1	pairwise $\wedge$	n^2	$(\alpha \cap \beta) - (\alpha' \cup \beta')$	n^2
$\cup$	$F_1 \vee F_2$	1	$F_1 \vee F_2$	1	$(\alpha \cup \beta) - [(\alpha' \cup \beta') -$ $((\alpha' \cap \beta) \cup (\alpha \cap \beta') - \alpha' \cap \beta')]$	n^2
$-$	$F_1 - F_2$	1		n^2	$F_1 - F_2$	1
$\pi_{\bar{x}}$	q. elim.	n^{2^l}	Fourrier-M.	$2^{2^l} n^2$		$2^{2^l} n^2$
$\times$	$F_1 \wedge F_2$	1	see $\cap$	n^2	$(\alpha \times \beta) - ((\alpha' \times \beta) \cup (\beta' \times \alpha))$	n^2
σ_F	$F_1 \wedge F$	1	see $\cap$	n	$\sigma_F(\alpha) - \sigma_F(\alpha')$	n
$\emptyset ?$		n^{2^k}		$2^{2^k} n^2$		$2^{2^k} n^2$
ν		n^{2^k}		$nlogn + n^{k/2}$		$n^2 + n^{k/2}$

Figure 2: Summary of the algebraic functions and their complexity.k is the dimension of the data and l the number of variables eliminated during a projection

3. $\sigma_F(F_1)$ is defined recursively by $\sigma_F(\alpha) - \alpha'$.
4. $F_1 \cup F_2$ is defined recursively by $(\alpha \cup \beta) - [(\alpha' \cup \beta') - ((\alpha' \cap \beta) \cup (\alpha \cap \beta') - \alpha' \cap \beta')]$.
5. $F_1 - F_2$ is defined recursively by $(\alpha - \alpha') - (\beta - \beta')$
6. $\pi_{\overline{x}}\ F_1$ is recursively calculated using an algorithm slightly more complex than the Fourier-Motzkin one but still quadratic when eliminating one variable. The output is $(\pi_{\overline{x}}\ \alpha) - \gamma$ where γ is obtained by computing the intersection points of α and β, and extracting those which are part of the boundary of γ.

The complexity of the operations differs from that of the algebra for the convex normal form (DNF). One can easily show that Union, Intersection, Cartesian product, and Projection are quadratic, while selction is linear and set difference is constant time.

Testing whether a relation is empty is the same as testing for $\alpha \subset \alpha'$ which can be done using negation over DNF formulae. It takes therefore time $O(n^2)$ in dimension 2 but is exponential in the dimension, in the general case. Normalization is defined as eliminating empty tuples and redundancies inside each tuple and, in each negation $\alpha - \beta$ restricting β so that $\beta \subset \alpha$. Normalization can be calculated in time $O(n^2)$ in dimension 2. For more information about the complexity we refer to [GO97, Sch86]

The data complexity of linear constraint queries has first been investigated in [KKR90], 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 [GS97]. It has also been shown in [GST94] 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).

3 Spatial data models and languages

In this section, we present the data model implemented in the DEDALE system, and the foundation for a spatial query language. The representation of spatial data is based on the logical framework described in the previous section.

Data is stored in non-first normal form relations [Ull88], to allow the manipulation of infinite sets of points as first class citizens. One level of set nesting is sufficient to achieve this goal. Spatial objects are typed objects, built from atomic objects (constants and rational numbers) using the set and tuple constructors. We assume the existence of an infinite set of attribute names Att, and consider two *atomic types*, rational, $\mathcal{Q}$, and constant, U. Let D be an infinite set of atomic constants of type U. The complex types and their domains are defined as follows.

1. $\{\mathcal{Q}^k\}$ is a *set type* (of arity k) with domain : $dom(\{\mathcal{Q}^k\}) = \wp_{fr}(\mathbb{Q}^k)$, where $\wp_{fr}(S)$ denotes the set of finitely representable subsets of S.

2. If $T_1, ..., T_n$ are atomic or set types, and $A_1, ..., A_n$ are attribute names, $[A_1 : T_1, ..., A_n : T_n]$ is a *tuple type* with domain : $dom([A_1 : T_1, ..., A_n : T_n]) = \{[A_1 : a_1, ..., A_n : a_n] | a_i \in dom(T_i)\}$.

3. If T is a tuple type, $\{T\}$ is a *relation type* with domain : $dom(\{T\}) = \wp_f(dom(T))$, where $\wp_f(S)$ denotes the set of finite subsets of S.

In the sequel, a *relation* will denote an object of some relation type, and a *set* an object of some set type. This distinction is fundamental. Note that relations are finite, while sets might be infinite. Data is strongly typed with the exception of non-nested *relations* with exclusively rational attributes that can be seen as *sets*.

For simplicity, in the type definition, we did not distinguish between distinct non-rational atomic types, such as string, color, etc, that are used in the examples. An example of a relation type for geography is ground-occupancy, GO.

$$GO = \{[\text{name} : string, \text{ground-type} : string, \text{owner} : string, \text{geometry} : \{\mathcal{Q}^2\}]\}$$

Relations admit a symbolic description which generalizes the first-order formulae of the previous section. The language contains distinct sorts of variables corresponding to the types U, $\mathcal{Q}$, and $\{\mathcal{Q}^k\}$. Variables v of type U occur in atoms of the form $v = a$, where $a \in D$, variables of type $\mathcal{Q}$ in linear constraints, and variables t of type $\{\mathcal{Q}^k\}$ in expressions of the form: $t = \{[x_1, x_2, ..., x_n] | \varphi(x_1, x_2, ..., x_n)\}$, where the x_i's are variables of type $\mathcal{Q}$.

An example of a spatial relation of type GO is given on figure 3.

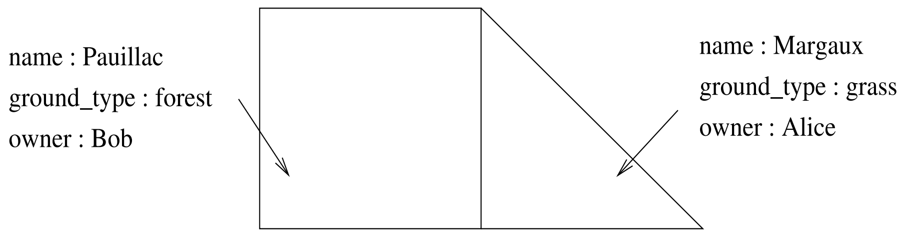

Figure 3: Example of GO

The symbolic representation of the previous relation (with the geometry in DNF) is given by the following table.

name	ground-type	owner	geometry
Pauillac	"forest"	Bob	$\{x \leqslant 1 \wedge x \geqslant -1 \wedge y \leqslant 1 \wedge y \geqslant -1\}$
Margaux	"grass"	Alice	$\{x \geqslant 1 \wedge y \geqslant -1 \wedge x + y - 2 \leqslant 0\}$

We now introduce the *dedale algebra* which extends relational algebra with operators for the specific types of dedale. The algebra contains a *Tuple* constructor $[o_1, \ldots, o_k]$ that builds tuples (of tuple type) from objects $o_1, \ldots, o_k$ of atomic or set types.

We distinguish between *valid* and *invalid* tuples. A tuple is *valid* if its attributes of set types are non empty sets, and invalid otherwise. Invalid tuples are discarded. The intuition behind this semantical choice relies on the fact that the space is stored in the sets, and an object with empty space is considered as void.

For each natural number i, there is an *attribute projection* $t.i$ that extracts the i^{th} attribute of a tuple, t, if t is valid, and is undefined otherwise. For example, if $t = [3, 0, \{1, 2\}]$, and $t' = [3, 0, \{\}]$, then $t.1 = 3$ while $t'.1$ is undefined. Note that the tuple projection is a non-trivial operation, which relies on a empty-set test. Testing the emptiness of the sets can require some computation. The empty tuple (i.e. tuple of arity zero) is denoted by $[\,]$. Sets (of set type) are constructed using the *set* operator, $\{\bar{x} | \varphi(\bar{x})\}$, where $\bar{x}$ is a tuple of rational variables, and $\varphi(\bar{x})$ is a formula. Sets can equivalently be constructed algebraically from $\mathbb{Q}^k$. Singleton relations are constructed using the *relation* constructor, $\{t\}$, from a tuple t.

The algebra contains the following Boolean constants. *True* is denoted by $\{\ [\,]\ \}$, and *False* by $\{\ \}$.

- *emptiness test*, $\emptyset^?$, is modeled by $\emptyset^?(R) = True$ if the relation or set R is empty, and $\emptyset^?(R) = False$. Note that because of the definition of *True* and *False*, $\emptyset^?$is idempotent over $\{\ \}$ and $\{\ [\,]\ \}$ (*True* and *False*).

- *membership test*, $\in^?$, defined between objects a of type T, and objects A of type $\{T\}$, is modeled by $a \in^? A = True$ if a is an element of A, and $a \in^? A = False$ otherwise.

Let $E(X)$ be any algebraic expression, with a variable X. Then $\lambda X.E(X)$ denotes a function. For instance, let X be a variable of type $[U, \{\mathcal{Q}^2\}]$, then

$$\lambda X.[X.1, \{[x, y] | [x - 1, y - 1] \in X.2\}]$$

denotes a function which performs a translation of the geometry.

The dedale algebra contains the following operations.

- *set operations*: *union*, $\cup$, *intersection*, $\cap$, and *set difference*, $-$, apply to pairs of inputs of the same *set* or *relation* type.

- *selection*, σ_F, applies to inputs of *relation* or *set* type. F is an atomic constraint over variables corresponding to the attributes given by name or position. This constraint is either of linear form (e.g. $4X + 3Y = 2$), or a set membership constraint (e.g. $X \in S$).

- *projection*, π, applies to inputs of *set* or *relation* type.

- *Cartesian product*, $\times$, applies to pairs of inputs of either both *set* types or both *relation* types.

- *Restructuring*, MAP applies to inputs of *relation* type. If $E(X)$ is an algebraic expression of tuple type T', with a tuple variable $X : T$, and R is a relation of type $\{T\}$, then $MAP_{\lambda X.E(X)}(R)$ defines the relation of type $\{T'\}$, in which each tuple t of R, has been replaced by $E(t)$.

- *unnest* applies to inputs of *relation* type with a nested attribute, and produces as output a set or a flat relation by unnesting each tuple (not always defined).

- *unionest*, applies to inputs of *relation* type with a nested attribute i. $unionest_i(R)$ replaces the tuples of R having the same values for all attributes other than i, by a single tuple having the same value for all attributes but i and for i the union of all the corresponding values of the i attribute.

Unless otherwise specified, the interpretation of the operations on inputs of set type corresponds to the semantics of the operations on linear constraint relations as presented in Section 2, and the interpretation on inputs of relation type corresponds to the semantics of classical relational algebra over finite relations. Note that the two semantics coincide at the abstract level.

We next consider a list of useful operations that are definable using the primitive operations introduced above.

- *nest* applies to inputs of *relation* or *set* type; its output is of *relation* type. The attributes which are to be nested have to be of type $\mathcal{Q}$ and independent[1] of the other attributes, and the result must be of finite cardinality (undefined otherwise). Let $I = \{i_1, ..., i_k\}$ be a set of such attributes in a relation R, and $J = \{j_1, ..., j_{k'}\}$ be the set of the other attributes of R.

$$nest_I(R) = MAP_{\lambda Y.[Y.j_1,...,Y.j_{k'}, \pi_I \sigma_{(x_j = Y.j)j \in J}(R)]}(R)$$

- *complement*, $compl_i(R)$, applies to inputs R of *relation* type. It replaces the attribute i which is of type $\{\mathcal{Q}^k\}$ by its complement. Let $J = \{j_1, ..., j_{k'}\}$ be the set of the other attributes of R.

$$compl_i(R) = MAP_{\lambda X[X.j_1,...,X.j_{k'}, \mathbb{Q}^k - X.i]}(R)$$

- *internest* is similar to *unionest* but computes the intersection of the attribute i values instead of the union for *unionest*.

$$internest_i(R) = compl_i(unionest_i(compl_i(R)))$$

For convenience, we also introduce a Boolean selection.

- *Boolean selection*, $\sigma^?_F$, applies to inputs of *relation* or *set* type. F is an atomic constraint. $\sigma^?_F(R) = \emptyset ?\emptyset ? \sigma_F(R)$.

The primitive operators introduced above realize the basic set operations over nested objects. They are powerful enough to express many useful spatial operations, but important topological and metric properties cannot be defined in this restricted setting.

We propose to include new operators in DEDALE with the following two requirements. First, the language must be *linear*, that is it should be closed for the relations we consider. Essentially, we

[1] A set of attributes I is independent from another set of attributes J in a relation or a set R, if $I \cap J = \emptyset$, and there is a formula φ representing R, such that no attribute of I occurs with an attribute of J in a constraint of φ.

have to preserve the linearity of the sets. Second the complexity of the queries should be tractable in practice. The following new operators respect those two requirements.

We first introduce the following two fundamental geometric operators in Q^2. Note that these operators can be generalized to higher dimensions.

- *axis*, $axis(\{p_1\}, \{p_2\}, \{p_3\})$, 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*, $median(\{p_1\}, \{p_2\}, \{p_3\})$, 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.

The language with the previous operators is linear. Interesting queries can be expressed such as the convex hull or the Voronoi diagram. The convex hull of a finite collection[2] of points $R : \{[\{\mathcal{Q}^2\}]\}$ is defined as follows.

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

where $triangle(X.1, X.2, X.3) = axis(X.1, X.2, X.3) \cap axis(X.1, X.3, X.2) \cap axis(X.3, X.2, X.1)$.

The Voronoi diagram of a finite collection of points $R : \{[\{\mathcal{Q}^2\}]\}$ is defined as follows.

$$internest_2 MAP_{\lambda X[X.1, median(X.1,X.2,X.1)]}(R \times R)$$

For metric properties, a great care is due to guaranty our two objectives of linearity and feasibility. Indeed, a circle around a point can easily be defined with an unrestricted use of distance. Moreover, unrestricted selection conditions with distance might express arbitrary polynomial conditions. Let $d : \mathbb{Q}^{2k} \rightarrow \mathbb{Q}$ be the Euclidean distance in $\mathbb{Q}^k$. We allow the following Boolean use of distance between two k-ary sets R and S which preserves linearity and feasibility.

- *distance*, $\sigma^?_{dist\Theta\alpha}(R, S) = True$ if $d(p_1, p_2)\Theta\alpha$ holds for some p_1 in R and some p_2 in S, and $\sigma^?_{dist\Theta\alpha}(R, S) = False$ otherwise, where Θ is a predicate and $\alpha \in \mathbb{Q}$.

Regions of the relation $R : \{[U, \{\mathbb{Q}^2\}]\}$ at distance α from a given region $r : \{\mathbb{Q}^2\}$ are defined as follows.

$$MAP_{\lambda X.[X.1, X.2 \times \sigma^?_{d \leqslant \alpha}(X.2, r)]}(R)$$

The previous example illustrates the use of the Boolean functions together with the invalid tuples. For each tuple $[t_1, t_2]$ of R, if the region t_2 is at distance α from r, then the tuple is mapped to itself, and otherwise to an invalid tuple $[t_1, \{\ \}]$.

Important topological queries are not expressible with the previous operations. Note that as long as linearity is concerned they can be freely added to the algebra. Indeed, the extension of the present algebra with any topological operator is linear. We therefore suggest the addition of an operator for connectivity.

[2]The nesting of relation R is adopted for uniformity. Spatial data are always stored in the imbricated sets.

- *connectivity*, $connect(R) = true$, if $R \subseteq \mathbb{Q}^k$ defines a topologically connected region, and $connect(R) = False$ otherwise.

It can be verified that the evaluation of DEDALE queries defined with all the operators introduced above can be done in polynomial time.

4 The User Query Language

This section presents the user query language of DEDALE, along with some examples of queries. This language meets two requirements. First it hides as much as possible the underlying complexity of the data model. In particular the rules related to typing restrictions and to the nested structure are transparent to the end-user. Second the language has a direct and easy translation towards the DEDALE algebra, thereby allowing the design of an optimizer generating efficient evaluations.

4.1 The syntax

Our language relies on an SQL-like syntax. It contains the following components:

1. Algebraic operations that operate on sets (i.e. spatial objects) (see Table 1).
2. The limited set of primitives which have been introduced in the DEDALE algebra to augment its expressive power (see Table 2).
3. Macro-operations which simplify the expression of the most common spatial data manipulations (see Table 3).

Note that macro-operations are simple shortcuts for complex algebraic expressions, while primitives may be considered as external functions with respect to the algebraic operators.

Table 1 gives the list of algebraic operators introduced in the language. We use the following notation: g_i is any attribute of set type $\{Q^k\}$, x_i is one of its projection, a_i is the i^{th} attribute of a relation R, $sort(R)$ is the set of attributes of R, and i_n denotes the n^{th} input argument.

Syntax	Algebraic equivalence	Type
g_1 INTER g_2	$g_1 \cap g_2$	$\{Q^k\}, \{Q^k\} \rightarrow \{Q^k\}$
g_1 UNION g_2	$g_1 \cup g_2$	$\{Q^k\}, \{Q^k\} \rightarrow \{Q^k\}$
g_1 CROSS g_2	$g_1 \times g_2$	$\{Q^{k_1}\}, \{Q^{k_2}\} \rightarrow \{Q^{k_1+k_2}\}$
g_1 MINUS g_2	$g_1 - g_2$	$\{Q^k\}, \{Q^k\} \rightarrow \{Q^k\}$
RESTRICT g WITH (F)	$\sigma_F(g)$	$\{Q^k\} \rightarrow \{Q^k\}$
PROJ g ON $(x_{i_1}, \ldots, x_{i_j})$	$\pi_{x_{i_1},\ldots,x_{i_j}}(g)$	$\{Q^k\} \rightarrow \{Q^j\}$
GROUP-UNION R ON a_i	$unionnest_i(R)$	$sort(R) \rightarrow sort(R)$
GROUP-INTER R ON a_i	$internest_i(R)$	$sort(R) \rightarrow sort(R)$
g IS NOT EMPTY	$\emptyset ? \emptyset ?(g)$	$\{Q^k\} \rightarrow bool$

Table 1: syntax of algebraic operations

In addition, we will use a Boolean version OP? of each algebraic operator, simply defined as OP(...) IS NOT EMPTY. We give now two tables listing respectively primitives and macros.

Name	Type	Comment
AXIS	$(\{Q^k\}, \{Q^k\}, \{Q^k\}) \rightarrow \{Q^k\}$	See Section 3
MEDIAN	$(\{Q^k\}, \{Q^k\}, \{Q^k\}) \rightarrow \{Q^k\}$	See Section 3
DIST	$(\{Q^k\}, \{Q^k\})\ COMP\ Q \rightarrow bool$	True if $dist(i_1, i_2)\ COMP\ Q$
CONNECT?	$(\{Q^k\}) \rightarrow bool$	See Section 3

Table 2: primitive operations

Name	Type	Comment
MIN, MAX	$(\{Q^k\}, x_i) \to Q$	Yields the min (resp. max) of i_1 with respect to the i^{th} variable
ADJACENT?	$(\{Q^k\}, \{Q^k\}) \to bool$	True if i_1 and i_2 are adjacent
TRIANGLE	$(\{Q^2\}, \{Q^2\}, \{Q^2\}) \to (\{Q^2\})$	i_1, i_2, and i_3 are single points Outputs the triangle
BOUNDARY	$(\{Q^k\}) \to (\{Q^k\})$	i_1 is a connected region R. Outputs the boundary of R
VERTICES	$(\{Q^k\}) \to (\{Q^k\})$	i_1 is a connected region R. Outputs the vertices of R

Table 3: macro-operations

It is easy to express MIN and MAX in the DEDALE alegra. The expression of TRIANGLE was given in section 3. The algebraic expression of the macro-operations ADJACENT, BOUNDARY and VERTICES can be found in [DGVG97].

Loosely speaking, the **select ... from ... where** structure can be seen as a convenient way to express the MAP operation. Invalid tuples are rejected in the **select ... from ... where** as in the MAP operation. Algebraic expressions on set of points can be freely introduced in each clause in order to define complex spatial queries. In that case, either the spatial attribute is manipulated as a whole using the attribute projection (e.g. $o.geo$), or the operation uses some particular variable (coordinate) which is simply denoted by $x_1, x_2, \ldots, x_k$.

4.2 Example of queries

The user language is now illustrated using a small sample of queries. There is no commonly accepted set a typical spatial queries. For our sample queries choice, we follow the classification of [Güt94], which distinguishes (i) *Spatial selection (point query, windowing, topological queries)*, (ii) *Spatial join*, (iii) *Spatial function application* (*clipping* for instance), and (iv) some other various set operations including *fusion*, *nearest neighbor*, etc. We also give more complex queries illustrating the ability of DEDALE to handle spatial-temporal and complex geometric operations.

For each query, we give both its expression in the DEDALE algebra, and in the user query language.

In the following the database schema is $\{GO, Cities, Waterpoints, Roads, Abbeys\}$, where GO stands for ground occupancy, each relation having the simplified schema $\{[name : U, geo : \{Q^2\}]\}$.

1. *Which parcel contains the point* (a, b) (spatial selection)

$$MAP_{\lambda X[X.1, X.2 \times \sigma^?_{x_1=a \wedge x_2=b}(X.2)]}(GO)$$

The $\sigma^?_{x_1=a\wedge x_2=b}$ expression simply denotes an emptiness test on the result of the σ operator: if the result of the selection is false, $\sigma^?$ returns an empty set, therefore the cross product with $X.2$ is also empty and the corresponding tuple is invalid.

select o.name, o.geo
from o **in** GO
where RESTRICT? o.geo WITH $(x_1 = a \wedge x_2 = b)$

Note that the expression of other spatial selections such as *windowing* would be similar.

2. *Give the part of the parcels in GO contained in a given rectangle* (clipping)

$$MAP_{\lambda X[X.1,X.2\cap rect]}(GO)$$

select o.name, o.geo INTER rect
from o **in** GO

3. *Parts of roads within a forest* (spatial join)

$$MAP_{\lambda X[X.1,X.2\cap X.4]}(Roads \times Forests)$$

select r.name, r.geo INTER f.geo
from r **in** Roads, f **in** Forests

4. *Parcels adjacent to a lake* (spatial join with a topological condition)

$$MAP_{\lambda X[X.1,X.2\times Adjacent?(X.2,X.4)]}(GO \times Lakes)$$

select o.name, o.geo
from o **in** GO, l **in** Lakes
where Adjacent? (o.geo, l.geo)

5. *Water-points south of Orange* (spatial join with a directional condition)

   The strategy is as follows: project Orange on its y-coordinate (denoted $yrange$), then take Water-points whose y-coordinate is less than at least one value in $yrange$.

$$MAP_{\lambda X[X.1,X.2\times\sigma^?_{x_1<x_2}(\pi_{x_2}(X.2)\times\pi_{x_2}(X.4))]}(Waterpoints \times \sigma_{name='Orange'}(Cities))$$

select w.name, w.geo
from w **in** Waterpoints, c **in** Cities
where c.name = 'Orange'
and RESTRICT?(PROJ (w.geo) ON (x_2)
CROSS
PROJ (c.geo) ON (x_2))
WITH ($x_1 < x_2$)

6. *Water-points less than 1KM from Orange*

$$MAP_{\lambda X[X.1,X.2\times\sigma^?_{dist<'1km'}(X.2,X.4)]}(Waterpoints \times \sigma_{name='Orange'}(Cities))$$

select w.name, w.geo
from w **in** Waterpoints, c **in** Cities
where DIST (w.geo, c.geo) < "1KM"
and c.name = 'Orange'

7. *Nearest water-point from each city*

   An intuitive solution to this problem is to select the water-point(s) whose distance with respect to city c is minimal by computing the distance between two points, a functionality which is not in the query language (see Section 3). However this query can be expressed in DEDALE as follows: given a city c, the nearest water-point w_1 is the one whose Voronoi polygon contains c, that is such that **for all** water-points w_2, c lies in the half-plane defined by the median of (w_1, w_2) and containing w_1.

$$MAP_{\lambda X.[X.1, X.3, X.4 \times \emptyset ? \emptyset ? (X.2 \cap internest_1(MAP_{\lambda Y.[median(X.4,Y.2,X.4)]}(Waterpoints)))]}(Cities \times Waterpoints)$$

```
select   c.name, w.name, w.geo
from     c in Cities, w in Waterpoints
where    c.geo INTER? GROUP-INTER (select MEDIAN(v.geo, w.geo, v.geo)
                                   from v in Waterpoints)
                      ON a1
```

   The above expression defines for each city, the Voronoi polygon associated to each water-point. A naive evaluation would be far from optimal. Fortunately, it is possible to obtain an efficient evaluation strategy thanks to the optimization rules presented in Section 5.

8. *Give roads that come across the roman abbeys region in Provence*

   By definition, a point is in the abbeys region as soon as it is contained by a triangle defined by three abbeys. The following expression defines the query.

$$\pi_1 MAP_{\lambda X.[X.1, X.2 \cap unionnest_1(MAP_{\lambda Y.[Triangle(Y.2,Y.4,Y.6)]}(Abbeys \times Abbeys \times Abbeys))]}(Roads))$$

```
select   r.name
from     r in Roads
where    r.geo INTER? GROUP-UNION(select Triangle(a1.geo, a2.geo, a3.geo)
                                  from  a1 in Abbeys,
                                        a2 in Abbeys,
                                        a3 in Abbeys)
                      ON a1
```

   Observe here again that a naive evaluation would compute the Abbeys convex hull for each road: once more one can use a better strategy by a clever rewriting of the query during the optimization phase.

9. *Give the boats which will be less than 10 KM from St Malo on Monday 22/09/97*

   This query as well as the following one is related to a spatio-temporal application. In this first example, objects are mobile, while in the following one the world is evolving. The query selects mobile objects, boats whose locations are changing with time. The schema of the relation *Boats* is $\{[name : U, traj : \{Q^3\}]\}$. The third component of the *traj* attribute denotes the time.

$$\pi_1 MAP_{\lambda X[X.1, \sigma^?_{dist<10km}(\pi_{x_1,x_2}(\sigma_{x_3='22/09/97'}(X.2)), X.4)]}(Boats \times \sigma_{name='St\,Malo'}(Cities))$$

select b.name
from b **in** Boats, c **in** Cities
where DIST (PROJ(RESTRICT b.traj WITH (x_3='22/09/97')) ON (x_1, x_2),
c.geo) < '10KM'
and c.name = 'St Malo'

10. *Display couples of harbors which are connected by a 15m-depth channel on 22/09/97 at 6pm.*

The sea is modeled as a 4-dimensional object. Its schema is $\{[name : U, shape : \{Q^4\}]\}$, the third coordinate of shape represents the depth and the fourth one the time.

$$\pi_{1,2}MAP_{\lambda X[X.3,X.5,Connect?(X.4 \cup X.6 \cup \pi_{x_1,x_2}(\sigma_{x_4='22/09/97:18' \wedge x_3>'15m'}(X.2)))]}(Sea \times Cities \times Cities)$$

select c1.name. c2.name
from s **in** Sea, c1 **in** Cities, c2 **in** Cities
where CONNECT? (c1.geo UNION
c2.geo UNION
PROJ (RESTRICT s.geo WITH($x_3 >' 15m' \wedge$
$x_4 =' 22/09/97 : 18'$))
ON (x_1, x_2))

Note that the CONNECT? primitive allows Boolean queries such as *Is this region connected?*, but not to output information regarding the connected components of a region.

11. *Given a region R with holes, and a source point S, give the subregions of R which are visible from S.*

This query is similar to the Art Gallery problem: given a room modeled by a non-convex polygon with holes, which part of the room is covered by a guard located at point p inside the polygon?

We assume that R and S are of set type $\{Q^2\}$. The set of vertices of R, as well as its boundary are definable in DEDALE. We assume that they have the same set type as R.

The strategy for computing the query consists in (i) drawing a straight line from S to every point in $vertices(R)$, (ii) computing the set I of intersection points between each straight line and the boundary of R and (iii) keeping the triangles created from S and couples of points in $I \cup vertices(R)$ that lie entirely in R. The algebraic expression is decomposed below. First we define the set I.

$$I(R,S) = unionnest_1 MAP_{\lambda X[axis(S,\{[X.1,X.2]\},\{[X.1,X.2]\}) \cap boundary(R)]} vertices(R)$$

Then compute the triangles. $I(R,S)$ is a finite set. Let $J(R,S) = I(R,S) \cup vertices(R)$.

$$Triangles(R,S) = MAP_{\lambda X[triangle(S,\{[X.1,X.2]\},\{[X.3,X.4]\})]}(J \times J)$$

Finally the visible subregions of R are computed as:

$$Visible(R,S) = MAP_{\lambda X[X.1 \times \emptyset?(X.1-R)]}(Triangles(R,S))$$

```
select   Triangle(s, p1, p2)
from     p1 in J, p2 in J
where    Triangle(s, p1, p2) Minus? R
```

Where J is defined by :

```
Vertices(R) UNION GROUP-UNION (select (Boundary(R) INTER AXIS(s,v,v)
                                  from v in Vertices(R)))
                  ON a_1
```

5 Query processing in DEDALE

In this section, we present the architectural framework of the currently implemented prototype, with a strong focus on the various modules involved in query optimisation and evaluation. We illustrate this core functionality upon some of the queries of Section 4. These examples show how logical expressions which express in a declarative way complex geometric operations can be translated towards an efficient evaluation.

Classical techniques tailored to spatial queries are exhibited: access path selection, query rewriting and translation from a logical expression to a physical query execution plan (QEP). We then introduce some more intricate queries. They naively express geometric operations which can be related to fundamental structures of computational geometry: convex hull and Voronoi diagrams. Thanks to rewrite rules and pattern recognition mechanisms, such queries can be identified in DEDALE, and thus processed efficiently.

5.1 Architecture

DEDALE has been implemented on top of the O_2 Database Management System [BDK92]. It relies as much as possible on O_2 existing modules, and in particular standard OQL evaluation is used for the part of queries on relational-like attributes. However, for the query evaluation mechanisms related to spatial data, DEDALE provides its own set of primitives, algorithms and access paths. Figure 4 illustrates the architecture of the prototype. It consists of the following components:

1. A Graphical User Interface (GUI) which allows to query the database with parameters such as points or rectangles, and to display query results including maps and other thematic data.
2. The DEDALE optimizer composed of the following modules: (i) the lexical and syntactical analyzer of a query which requires the access to the data dictionary stored in the O_2 database, (ii) the semantic analyzer which transforms an algebraic expression into a near-optimal query execution plan.
3. The query processor. It controls the split of individual operations in the QEP as follows: atomic attributes are processed through standard OQL evaluation, while spatial data is retrieved *via* multi-dimensional access paths and processed by functions of the constraint engine module.
4. The O_2 DBMS providing data storage and schema information.

Except for the optimizer which is under implementation, all the components of this architecture are already available. In the current implementation, query processing relies on OQL standard evaluation. The first objective was to validate the choice of a representation with constraints as

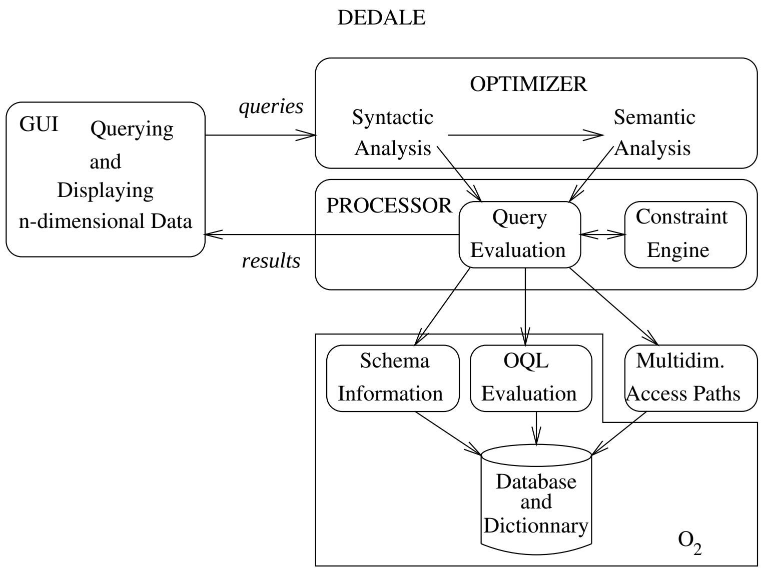

Figure 4: Architecture of DEDALE

well as the primitives for constraints manipulation which are carried out through function calls in OQL. This work has been described in [GRSS97] where a detailed description for the various algorithms involved in constraint manipulation were given.

Since OQL is not aware of the specific semantics of DEDALE algebraic operators, the current implementation yields non-optimal processing for queries that mix atomic and point-sets attributes. In addition, the lack of spatial indexes in O_2 makes it difficult to tailor the standard OQL evaluation for DEDALE queries. A new query optimizer, implementing the evaluation principles presented below, will supply an adequate tool for an efficient processing of queries in DEDALE.

5.2 Acces paths and physical primitives

DBMSs should provide access paths to data. This is usually fulfilled by physical access schemes such as B-trees or hashing provided by the system. For multi-dimensional data, many structures have been proposed, such as the R-Tree [Gut84] and its variants [SRF87, BKSS90]. Unfortunately, most of these structures rely on efficient clustering mechanisms, a feature which is difficult to handle in DEDALE since we build the system on top of the O_2 system.

Therefore we are currently evaluating two different schemes. First a variant of the R-tree, the R*Tree [BKSS90], has been implemented and stored as an O_2 object. Clustering, although not completely satisfactory, is partially obtained through some heuristics during insertion. The second approach follows [SGR96]. It proceeds in two phases: a simple grid [NHS84] is first constructed, then linearized. The linear ordered cells are then automatically structured as an O_2 B-Tree. Such an approach, advocated in numerous papers [AS83, Sam90, Gae95], allows to take advantage of the DBMS native clustering mechanisms and possibly of some of the optimization and concurrency control provided by the DBMS.

Spatial indexes behave slightly differently from data structures used in classical databases: while a B-Tree for instance allows an exact predicate matching, a spatial index is constructed on an approximate representation of geometry (usually the bounding box) and involves a two-steps processing: the index traversal is a *filtering step* which delivers a superset of the result. It must be followed by a *refinement step* which evaluates the predicate over the exact spatial representation of each object. This behavior is represented in optimisation rules related to set of points in the Appendix.

Next we consider the physical operators that can be used to evaluate a logical algebraic expression. They are quite similar to those commonly found in classical relational databases [Gra93], the main differences arising from the particular semantics of operators applied to set of points.

1. Predicates used to filter data are subject to **lazy evaluation**. The result, which must comply with the constraint representation, is in this case a simple concatenation of the operations input(s). The result might be inconsistent for some tuples which are thereby invalid.

2. As a consequence, it may be necessary to carry out an intermediate evaluation at some step of the query execution plan (QEP), depending on the laziness of the operations performed in the subtree.

We denote by *normalization* this evaluation. Normalization eliminates redundancy and detects inconsistencies: if the resulting set is empty, then the tuple is unsatisfiable. The normalization algorithm works then as follows: first the constraints are scanned in order to count the number of constraints with predicate *equal*. Except for degenerated cases, this number indicates the type of the result: if it is larger than 2, the tuple is non satisfiable; if it is equal to 2, the constraint represents a point, 1, a linear object and 0 means that every constraint defines a half-plane, in which case *half-plane intersection* [PS85] computes a convex polygon which is converted into constraints.

Choosing the right place in the execution plan to carry out the normalization process is an essential part of the query processing strategy, and still an open issue. There are at least three motivations to perform a normalization: (i) to remove invalid tuples from a final result when lazy operations have been performed, (ii) to evaluate emptiness of a set of points before projecting out this set and (iii) to reduce the size of intermediate results. While motivations (i) and (ii) are easily introduced as optimisation rules and illustrated below, the latter is highly dependent on cost models, a topic which goes beyond the scope of this paper.

Given available access paths and physical primitives, the optimiser should rely on algebraic equivalences and translation rules to convert a logical query towards an efficient query execution plan. We illustrate the main principles in the remainder of this section, in the style of [Fre87], upon a sample of representative queries. The technical material is given in the Appendix

5.3 Evaluation of sample queries

The next examples shows that, given the declarative language of DEDALE, classical optimisation strategies can easily be extended to handle spatial data as well. It also highlight the issue of tuples normalization during the execution of a query plan. This topic is quite specific to constraint database management, and the impact of normalization strategies on QEP efficiency deserves careful analyses and performance evaluations.

In the following, each relation has for schema $\{[name : U, geom : \{Q^2\}]\}$. It is assumed that an index I_{geom} exists on the $geom$ attribute.

1. *Which parcels intersect the rectangle rect* (windowing). The logical expression is

$$MAP_{\lambda X[X.1, X.2 \times \sigma^?_{rect}(X.2)]}(GO)$$

First emptiness tests are removed by using rewrite rule 1b (see Appendix). One considers also access paths. We have two solutions: a full scan or the spatial index traversal. One obtains:

1. $\pi_{1,2}(MAP_{\lambda X[X.1,X.2,\sigma_{rect}(X.2)]}(Fscan(GO)))$
2. $\pi_{1,2}(MAP_{\lambda X[X.1,X.2,\sigma_{rect}(X.2)]}(Iscan(I_{geom},<>)))$

Now we push predicates inside the index traversal (rule **TRsel4** in Appendix), and keep only those expressions which use as many index as possible (i.e. expression a). MAP is converted to its physical equivalent, $Restruct$. The physical operation for σ can be $Filter^{norm}$ or $Filter^{lazy}$ (rule **TRsel2**). This yields two solutions for expression (a) above:

. $\pi_{1,2}(\underbrace{Restruct(Daddress(Iscan(I_{geom},< rect >)), < 1 : 2 : Filter^{lazy}(rect, 2) >)}_{e_a})$

. $\pi_{1,2}(\underbrace{Restruct(Daddress(Iscan(I_{geom},< rect >)), < 1 : 2 : Filter^{norm}(rect, 2) >)}_{e_b})$

Climbing up the tree, the next operation is the projection. Since, in expression (a), $\pi_{1,2}$ projects out set attributes and e_a results from a lazy operation, one must first normalize e_a (rule **TRproj2**). In expression (b), e_b is already normalized and a simple relational projection suffices (rule **TRproj3**)

1. $Proj^{rel}(< 1, 2 >, Norm(e_a))$.
2. $Proj^{rel}(< 1, 2 >, e_b)$.

The two candidate QEP differ by the position of the $Norm$ operation in the query tree: in e_a, a lazy selection is performed first on the result of the index traversal. The normalization is then carried out in a second scan. In e_b, the constraint concatenation and its evaluation are processed in the same scan. Therefore, (b) is probably better since it spares one scan.

This might not be always the case. Suppose now that the query is *Give parcels named "Foo" that intersect rect*. We would have to choose between the two following QEP:

1. $Proj^{rel}(< 1, 2 >, Norm(Filter^{rel}('Foo', e_a)))$.
2. $Proj^{rel}(< 1, 2 >, Filter^{rel}('Foo', e_b))$.

In that case, it is probably better to process the relational selection **before** the normalization since it is far less costly. (a) is probably the best choice.

2. *Give the part of Orange contained in this rectangle* (clipping).

$$MAP_{\lambda X[X.1,X.2 \cap rect]}(\sigma_{name='Orange'}(Cities))$$

We first consider the available access paths, and translate MAP and $\cap$ operations.

1. $\sigma_{name='Orange'}(Restruct(Fscan(Cities), < 1 : Join(p_\cap, rect, 2) >))$
2. $\sigma_{name='Orange'}(Restruct(Iscan(I_{geom},<>), < 1 : Join(p_\cap, rect, 2) >))$

The bounding box of $rect$ (i.e. $rect$ itself) can be used for index traversal (rule **TRjoin5**) in expression (b). The refinement step of the $\cap$ operation can be performed either in lazy or normalized mode.

1. $\sigma_{name='Orange'}(Restruct(Fscan(Cities), < 1 : \cap_{nl}^{mode}(rect, 2) >)$

2. $\sigma_{name='Orange'}(Restruct(Iscan(I_{geom}, < rect >), < 1 : \cap_{nl}^{mode}(rect, 2) >)$

In a third step, one applies translation rules for selection. Selection is pushed down the QEP in expression (a) (rule **TRsel1**).

1. $Restruct(Filter^{rel}('Orange', Fscan(Cities)), < 1 : \cap_{nl}^{mode}(rect, 2) >)$
2. $Filter^{rel}('Orange', Restruct(Iscan(I_{geom}, < rect >), < 1 : \cap_{nl}^{mode}(rect, 2) >)$

It is probably better to postpone the normalization after the relational selection: $mode$ is set to $lazy$. This yields, as a result of the whole expression, a set on non-normalized tuples. Therefore a final $Norm$ operation has to be performed. One finally gets:

1. $Norm(Restruct(Filter^{rel}('Orange', Fscan(Cities)), < 1 : \cap_{nl}^{lazy}(rect, 2) >))$
2. $Norm(Filter^{rel}('Orange', Restruct(Iscan(I_{geom}, < rect >), < 1 : \cap_{nl}^{lazy}(rect, 2) >)))$

It remains to evaluate the cost of the two expressions to choose the QEP. (b) is probably the best one since it saves a full scan.

3. *Part of roads within a forest* (spatial join).

$$MAP_{\lambda X[X.1, X.2 \cap X.4]}(Roads \times Forests)$$

We concentrate on QEP featuring spatial indices. This limits the choices to three solutions:

1. $Restruct(Iscan(I_{roads}, <>) \times Fscan(Forests), < 1 : Join(p_{\cap}, 2, 4) >)$.
2. $Restruct(Iscan(I_{roads}, <>) \times Iscan(I_{forest}, <>), < 1 : Join(p_{\cap}, 2, 4) >)$.
3. $Restruct(Fscan(Roads) \times Iscan(I_{forest}, <>)), < 1 : Join(p_{\cap}, 2, 4) >)$.

Choices (a) (modulo exchange of join arguments) and (c) are relevant since, for simplicity, we do not consider synchronized tree traversal as suggested in (b). We focus on the development of expression (c): the bounding box of each road tuple can be used as a criteria for accessing the forests collection through the index (rule **TRjoin5**). One obtains:

$$Restruct(Join(<>, Fscan(Roads), Iscan(I_{forest}, bb(road.geo))), < 1 : \cap_{nl}^{mode}(2, 4)) >)$$

The final QEP follows the classical scan-and-index strategy: scan the $Roads$ collection, access to $Forest$ through the spatial index, using as a traversal criteria the bounding box of the spatial attribute of the current tuple in $Roads$. Finally perform the intersection. Depending on $mode$, a final normalization operation should be processed to conclude the execution.

5.4 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.

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 for instance 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 complexity of the naive evaluation of this query is in $O(n^3)$, a rather bad complexity compared to the $O(n\log(n))$ complexity of efficient algorithms based on a divide and conquer strategy [PS85].

If we are able to follow a divide and conquer strategy we should improve the query evaluation as well. This is what permits the following generic pattern. It defines the convex hull as a union of triangles constructed over a set of points.

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

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

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

It remains to show how we can identify ER1 patterns when rewriting queries. We illustrate this with the example of *Roman abbeys in provence*, using the rules given in the Appendix.

$$\pi_1 MAP_{\lambda X.[X.1, X.2 \cap unionnest_1(MAP_{\lambda Y.[Triangle(Y.2,Y.4,Y.6)]}(Abbeys \times Abbeys \times Abbeys))]}(Roads))$$

The naive evaluation would result in a $O(n \times m^3)$ complexity, where $n = |Roads|$ and $m = |Abbeys|$. From rule 1c in the Appendix, one obtains:

$$\pi_1 MAP_{\lambda X[X.1, X.2 \cap X.3]}(\underbrace{MAP_{\lambda Y[Y.1, Y.2, unionnest_1(MAP_{\lambda Z.[Triangle(Z.2,Z.4,Z.6)]}(Abbeys \times Abbeys \times Abbeys))]}(Roads))}_{A}$$

Rule 2b holds for A which becomes:

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

Finally, by rules 1d and 2a one obtains:

$$MAP_{\lambda Y.[Y.1,Y.2]}(Roads) \times \underbrace{unionnest_1(MAP_{\lambda X.[Triangle(X.2,X.4,X.6)]}(Abbeys \times Abbeys \times Abbeys))}_{B}$$

Part B 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.

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

and

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

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 tp 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 the optimization of the nearest neighbor query given in Section 4.

$$MAP_{\lambda X.[X.1,X.3,X.4\times\emptyset^?\emptyset^?(X.2\cap internest_1(MAP_{\lambda Y.[median(X.4,Y.2,X.4)]}(Waterpoint)))]}(Cities \times Waterpoint)$$

One applies rules 1b and 1c in order to extract the $\emptyset^?\emptyset^?$ and the $\cap$ operators. One obtains:

$$\pi_{1,2,3}MAP_{\lambda X.[X.1,X.3,X.4,X.2\cap X.5]}(\underbrace{MAP_{\lambda Y.[Y.1,Y.2,Y.3,Y.4,internest_1(MAP_{\lambda Z.[median(Y.4,Z.2,Y.4)]}(Waterpoint))]}(Cities \times Waterpoint)}_{A})$$

Now by rule 2b, A is equivalent to:

$$internest_5(\underbrace{MAP_{\lambda Y.[Y.1,Y.2,Y.3,Y.4,median(Y.4,Y.6,Y.4)]}(Cities \times Waterpoint \times Waterpoint)}_{B})$$

The commutativity between MAP and $\times$ holds for B (rule 1d) which is equivalent to:

$$\underbrace{MAP_{\lambda Y.[Y.1,Y.2]}(Cities) \times MAP_{\lambda Z[Z.1,Z.2,median(Z.2,Z.4,Z.2)]}(Waterpoint \times Waterpoint)}_{C}$$

Finally, according to rule 2a, the expression $internest_5(C)$ is equivalent to:

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

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**). Then computing the nearest neighbor query is optimal, i.e. of the order of $\Theta(n \log n)$: first the Voronoi diagram (n polygons) is computed in $n \log n$, then the second MAP operation scans the Voronoi collection and executes a point query. The operation is usually in $O(n)$, but it is well known (Euler's formula), that the average number of edges of a Voronoi polygon is bounded by six. Therefore the second MAP, corresponding to the point-query operation is processed in $O(6 \times n) = O(n)$

6 Conclusion

The DEDALE prototype, at this stage of development demonstrates already that it is possible to design spatial database systems with (i) a clear separation between logical and physical levels, allowing (ii) the design of declarative user-friendly query languages, with a formal semantics, which in turn allows the development of powerful optimization techniques. The prototype is implemented, and runs on two sets of geographic applications.

The optimization of spatial queries expressed in a declarative language has to the best of our knowledge not been addressed yet in the literature. We have demonstrated here that it is a promising issue. The complex queries involving cascade of geometric operations can be rewritten in queries leading to optimal evaluation, with standard algorithms of computational geometry.

The design of high level user-friendly query languages for spatial data depends from the success in practice of the optimization of declarative spatial queries. The optimization module of DEDALE is currently under implementation, and will be checked on a large benchmark of applications involving spatial reasoning.

Acknowledgments

The authors wish to warmly thank Michel Scholl and Serge Abiteboul for their careful reading of the present manuscript. Moreover we are greatly indebted to Michel who contributed from the beginning to the DEDALE system, and initiated many of its features.

References

[AS83] D. Abel and J.L. Smith. A Data Structure and Algorithm Based on a Linear Key for a Rectangle Retrieval Problem. *Computer Vision, Graphics and Image Processing*, 24:1–13, 1983.

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

[BCD89] F. Bancilhon, S. Cluet, and C. Delobel. A Query Language for an Object-Oriented Database System. In *2nd Int. Worshop on Database Programming Languages (DBPL)*, pages 301–322, 1989.

[BDK92] F. Bancilhon, C. Delobel, and P. Kanellakis, editors. *Building an Object-Oriented Database System: The Story of* O_2. Morgan Kaufmann, San Mateo, California, 1992.

[BKSS90] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*tree : An Efficient and Robust Access Method for Points and Rectangles. In *Proc. ACM SIGMOD Intl. Symp. on the Management of Data*, pages 322–331, 1990.

[Cod70] E.F. Codd. A relational model of data for large shared data banks. *Communications of ACM*, 13:6:377–387, 1970.

[DGVG97] F. Dumortier, M. Gyssens, L. Vandeurzen, and D. Van Gucht. On the decidability of semi-linearity for semi-algebraic sets and its implications for spatial databases. In *Proc. ACM Symp. on Principles of Database Systems*, 1997.

[Fre87] J. C. Freytag. A rule-based view of query optimization. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 173–180, San Francisco, 1987.

[Gae95] V. Gaede. Optimal Redundancy in Spatial Databases Systems. In *Intl. Conf. on Advances in Spatial Databases, (SSD'95)*, pages 96–116. Springer Verlag LNCS 951, 1995.

[GK97] S. Grumbach and G. Kuper. Tractable recursion over geometric data. In *International Conference on Constraint Programming*, 1997.

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

[Gra93] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[GRSS97] S. Grumbach, P. Rigaux, M. Scholl, and L. Segoufin. Dedale: A spatial constraint database. In *Intl. Workshop on Database Programming Languages (DBPL'97)*, 1997.

[GS95] R.H. Güting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *The VLDB Journal*, 4(3):243–286, 1995.

[GS97] S. Grumbach and J. Su. Queries with arithmetical constraints. *Theoretical Computer Science*, 173, 1997. Invited to a special issue.

[GST94] S. Grumbach, J. Su, and C. Tollu. Linear constraint query languages: Expressive power and complexity. In D. Leivant, editor, *Logic and Computational Complexity*, Indianapolis, 1994. Springer Verlag. LNCS 960.

[Gut84] A. Guttman. R-trees : A Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD Intl. Symp. on the Management of Data*, pages 45–57, 1984.

[Güt89] R.H. Güting. Gral: An Extensible Relational Database System for Geometric Applications. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, 1989.

[Güt94] R.H. Güting. An Introduction to Spatial Database Systems. *The VLDB Journal*, 3(4), 1994.

[Her96] J. Herring. The ORACLE 7 Spatial Data Option. Technical report, ORACLE Corp., 1996.

[Ja97] J.Patel and al. Building a scalable geo-spatial dbms : Technology, implementation, and evaluation. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 1997.

[KG94] P. Kanellakis and D. Goldin. Constraint programming and database query languages. In *Manuscript*, 1994.

[KKR90] P. Kanellakis, G Kuper, and P. Revesz. Constraint query languages. In *Proc. 9th ACM Symp. on Principles of Database Systems*, pages 299–313, Nashville, 1990.

[KPV95] B. Kuijpers, J. Paredaens, and J. Van den Bussche. Lossless representation of topological spatial data. In M. J. Egenhofer and J. R. Herring, editors, *Advances in Spatial Databases, 4th Int. Symp., SSD'95*, pages 1–13. Springer, 1995.

[KY85] D. Kozen and C. Yap. Algebraic cell decomposition in np. In *Proc IEEE Foundations of Computer Science*, pages 515–521, 1985.

[Mor89] S. Morehouse. The Architecture of ARC/INFO. In *Proc. Intl. Symp. on Computer-Assisted Cartography (Auto-Carto 9)*, pages 266–277, 1989.

[NHS84] J. Nievergelt, H. Hinterger, and K.C. Sevcik. The Grid File: An Adaptable Symmetric Multikey File Structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984.

[OM88] J. Orenstein and F. Manola. PROBE: Spatial Data Modeling and Query Processing in an Image Database Application. *IEEE Transactions on Software Engineering*, 14(5):611–628, 1988.

[PS85] F. Preparata and M. Shamos. *Computational Geometry: An Introduction.* Springer Verlag, 1985.

[PVV94] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proc. 13th ACM Symp. on Principles of Database Systems*, pages 279–288, 1994.

[RFS88] N. Roussopoulos, C. Faloutsos, and T. Sellis. An Efficient Pictorial Database System for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639–650, 1988.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley Publishing Company Inc, 1990.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1986.

[SFGM93] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The Sequoia 2000 Benchmark. In *Proc. ACM SIGMOD Intl. Symp. on the Management of Data*, 1993.

[SGR96] M. Scholl, G. Grangeret, and X. Rehse. Point and window queries with linear spatial indices: An evaluation with O_2. Technical Report RRC-96-09, Cedric Lab, CNAM, Paris, 1996. Available at http://sikkim.cnam.fr.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+Tree: A Dynamic Index for Multi-Dimensional Objects. In *Proc. Intl. Conf. on Very Large Data Bases (VLDB)*, pages 507–518, 1987.

[SV89] M. Scholl and A. Voisard. Thematic Map Modeling. In *Proc. Intl. Symp. on Large Spatial Databases (SSD)*, LNCS No. 409, pages 167–192. Springer-Verlag, 1989.

[Tom90] C. D. Tomlin. *Geographic information Systems and Cartographic Modeling.* Prentice-Hall, 1990.

[Ube94] M. Ubell. The Montage Extensible DataBlade Architecture. In *Proc. ACM SIGMOD Intl. Conference on Management of Data*, 1994.

[Ull88] J.D. Ullman. *Database and Knowledge Base Systems.* Computer Science Press, 1988.

Appendix

This appendix presents the technical rules related to the evaluation of queries in DEDALE. We focus on the manipulation of poinsets specific to DEDALE. First we give the main equivalence rules among algebraic expressions. They allow the optimizer to reorganize a query in order to exploit useful patterns for subsequent evaluation. Next we describe the physical language used in the current implementation of DEDALE. Finally translation rules from algebraic expression to physical operation are defined.

The whole constitute the basis of the query evaluation and optimization module in DEDALE. This specification should be completed by two components: cost models for the various operations involved in query processing, and a search strategy defining some heuristics to prune QEP which are likely to be too costly. These topics are subject to future work.

6.1 Rewriting rules

Rewriting rules (RR) include all the classical rules occurring in relational databases (which are hence omitted in the sequel), plus rewriting rules related to the behavior of the operators of DEDALE that operate on sets of points.

1. **RR for the restructuring operator (MAP)**

   (a) **Equivalence of MAP and π**

   $$MAP_{\lambda X.[X.1,\ldots,X.n]}(R) = \pi_{1,\ldots,n}(R)$$

   (b) **Removing emptiness tests inside MAP**

   $$MAP_{\lambda X.[X.1,\ldots,X.i \times \emptyset ? \emptyset ? (op(X.n_1,\ldots X.n_j))]} \equiv \pi_{1,\ldots,i} MAP_{\lambda X.[X.1,\ldots,X.i, op(X.n_1,\ldots X.n_j)]}$$

   In words: compute separately the operation *op* as an additional attribute a_{i+1}, then project out a_{i+1}. During projection, the validity of the tuple (and therefore the emptiness of a_{i+1}) must be checked (see translation rules below).

   (c) **Cascade of restructuring**

   $$MAP_{\lambda X.[X.1,\ldots,X.i, alg_1(X.n_1,\ldots X.n_j) o alg_2(X.m_1,\ldots X.m_k)]}(R)$$
   $$\equiv$$
   $$MAP_{\lambda X.[X.1,..,X.i, X.i+1 o alg_2(X.m_1,..,X.m_k)]}(MAP_{\lambda X.[X.1,..,X.i, alg_1(X.n_1,..,X.n_j), X.m_1,..,X.m_k]}(R))$$

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

   (d) **Commuting MAP and $\times$**

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

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

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

   (e) **Commuting MAP and σ.**

   $$MAP_{\lambda X.[X.1,\ldots,X.n]}(\sigma_F(R)) \equiv \sigma_F(MAP_{\lambda X.[X.1,\ldots,X.n]}(R))$$

   if F addresses only attributes in $sort(MAP) \cap sort(R)$.

2. **Rules for special operators**

(a) **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.

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

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

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

6.2 The physical algebra

The main specificity regarding query evaluation is the laziness of constraints manipulation. We introduce two physical operations to deal with this specific feature:

- *Norm(e)* normalizes the output of e.
- *Lazy(e)*, boolean, is true if the two following conditions are met: (i) there exists one branch b of the subtree which contains a lazy operation for a node n_l, and (ii) b does not contain a $Norm$ node n_n such that $n_n < n_l$.

Lazy(e) is true if the output of e is not yet evaluated, in which case it may contain (not yet detected) invalid tuples.

Operations which apply this symbolic manipulation are marked with the ^{lazy} exponent. Operations which carry out an actual evaluation are marked with the ^{norm} exponent. Note that an op^{norm} operation **is not equivalent** to $op^{lazy} \circ Norm$ since the evaluation of the predicate in carried out "on the fly" during the scan of the input in the former case, while this evaluation is processed in two successive scans in the latter one. In addition, we use the following notation

- $T(e)$ is the set of relation's names in e. For convenience, we extend the notation to handle predicates p as well: $T(p)$ is the set of relation's names occurring in p.
- $sort(e)$ is the set of attributes in e. In addition:
  1. $sch_{rel}(e)$ is the set of alphanumeric attributes in e (i.e. attribute whose domain is U).
  2. $sch_{geom}(e)$ is the set of attributes of e whose type is $\{Q^k\}$.
- $Ind(R, I_{\vec{A}})$ is true if I is an index on R built on $\vec{A}$, $\vec{A}$ being a vector over $sort(R)$.
- $Set(e)$ is true if e is a set, $Relation(e)$ is true is e is a relation.

Access Paths. We consider three types of accessing a relation: full scan, index scan and direct access to a tuple using its address.

- $Fscan(r)$ is a full scan of r. It delivers the full set of unordered tuples.
- $Daccess(< addr >, r)$ is a direct access to r using the list of addresses in $< addr >$.
- $Iscan(I, < ip >)$ is an index traversal using the $< ip >$ predicates. It delivers a set of tuple's addresses.

Each logical operator is now considered in turn, and its associated set of equivalent physical operations is given. The algorithms can be found in [GRSS97].

Restructuring, MAP. MAP applies to its input a set of algebraic operations. The physical operation is simply denoted as:

- $Restruct(e, < op_1 : op_2 : \ldots : op_n >)$

The laziness of the result is defined as $lazy(op_1) \vee lazy(op_2) \ldots \vee lazy(op_n)$.

Selection, σ. We distinguish relational selection which directly evaluates a criteria from the constraint selection which involves more complicated algorithms. Relational-like evaluation is never lazy.

- $Filter^{rel}(F, e)$ is the relational selection (direct evaluation of F upon each tuple of e).
- $Filter^{lazy}(F, e)$ is the lazy selection (i.e., make the conjunction of F with each tuple in e).
- $Filter^{norm}(F, e)$ add F to each tuples in e and normalizes "on the fly" in order to remove empty tuples.

Projection, π.

1. $Proj^{rel}(\vec{A}, e)$ is the relational projection.
2. $Proj^{dim2}(\vec{A}, e)$ is the peculiar algorithm used in dimension two: normalize first, and then project the bounding box on the appropriate axis. The result is normalized.
3. $Proj^{FM}(\vec{A}, e)$ is the Fourier-Motzkin projection. It takes as an input a (possibly non-normalized) relation and return a non-normalized relation after variable elimination.
4. $Proj^{norm}(\vec{A}, e)$ applies Fourier-Motzkin to each tuple and normalizes it.

Join, $\bowtie$. For simplicity, we consider only three usual join methods: (i) nested loop, (ii) scan-index (i.e. a nested loop with an index traversal to access the inner relation) and (iii) sort-merge. This leads to the following physical operations[3].

1. $Join^{rel}_{sm}(p, e_1, e_2)$ is the relational merge join. e_1 and e_2 are assumed to be sorted on the attributes in $sort(p)$. If needed, a sort operation is first carried out.
2. $Join^{rel}_{nl}(p, e_1, e_2)$ is the relational nested-loop join.
3. $Join^{lazy}_{nl}(p, e_1, e_2)$ is a nested-loop join with lazy evaluation.
4. $Join^{norm}_{nl}(p, e_1, e_2)$ is a nested-loop join with normalized evaluation.
5. $\cap^{lazy}_{nl}(e_1, e_2)$ is a join between two relations of same sort (i.e. $sort(e_1) = sort(e_2)$). The algorithms differs from the more general join algorithm by the bounding boxes overlapping filter.
6. $\cap^{norm}_{nl}(e_1, e_2)$ is the same as previously, except for the evaluation of each tuple as soon as it is produced.

Union, $\cup$

1. $\cup_{dm}(R, S)$ is the union with duplicate removal.
2. $\cup^{lazy}(R, S)$ is the union without duplicate removal.

Difference, $-$

1. $DIFF^{rel}_{nl}(R, S)$ is the nested-loop relational difference.
2. $DIFF^{rel}_{sm}(R, S)$ is the nested-loop sort-merge difference.
3. $DIFF_{geom}(R, S)$ is the geometric difference (see [GRSS97]).

[3]The $Join(p, e_1, e_2)$ is a generic notation which denotes the join of tuples from e_1 with tuples from e_2 according to the predicate p. This notation covers the cross-product as well: it can be seen a special join with empty predicate.

6.3 Translation rules

The following rules express the translation from an algebraic expression from the logical level to the physical level, in the style of [Fre87].

Access Paths

1. $R \rightarrow Fscan(R))$.
2. For each I such that $\exists \vec{A}, Ind(R, I_{\vec{A}})$, $R \rightarrow Iscan(I, <>)$.

Restructuring, $MAP(e)$. The rule is simply:

$$MAP_{\lambda X.[op_1(X),\ldots,op_n(X)]}(e) \rightarrow Restruct(e, < op_1 : op_2 : \ldots : op_n >)$$

Selection, $\sigma_F(e)$. Depending on the type of e, relation or set, the relevant algorithm is applied. If e is a set, then the selection appears within a MAP restructuring. Whenever possible, the index scan predicates are augmented with F.

1. **TRsel1** (Push down relational selections). If $Relation(e) \wedge e = (...Access(R))$, $Access \in \{Fscan, Daccess\}$, then
$$\sigma_F(e) \rightarrow (...Filter^{rel}(F, Access(R))).$$
2. **TRsel1'** (Merge relational selections). If $Relation(e) \wedge e = (...Filter^{rel}(p, Access(R))))$, $Access \in \{Fscan, Daccess\}$, then
$$\sigma_F(e) \rightarrow (...Filter^{rel}((F \wedge p, Access(R)))).$$
3. **TRsel2** (Rules for set attributes). If $Set(e)$, then
$$MAP_{\lambda X.[\ldots\sigma_F(e)\ldots]}(R) \rightarrow MAP_{\lambda X.[\ldots Filter^{mode}(F,e)\ldots]}(R), mode \in \{lazy, norm\}.$$
4. **TRsel2'** (Merge set-based selection, keeping the same mode). If $Set(e) \wedge e = Filter^{mode}(p, ...)$, then
$$MAP_{\lambda X.[\ldots\sigma_F(e)\ldots]}(R) \rightarrow MAP_{\lambda X.[\ldots Filter^{mode}(F \wedge p,e)\ldots]}(R)$$
5. **TRsel3** (Use F as relational index traversal criteria). If $Relation(e) \wedge e = (...Iscan(I, l_I))$ with $Ind(R, I_{\vec{A}})$ and any permutation of $sort(l_I) \cup sort(F)$ is a prefix of $\vec{A}$.
$$\sigma_F(e) \rightarrow (...Iscan(I, l_I \wedge F))$$
6. **TRsel4** (Use F as spatial index traversal criteria). If $Set(e)$ and e appears in an expression of the form $MAP_{\ldots\sigma_F(e)\ldots}(...Iscan(I_G, l_I))$ where I_G is a spatial index on attribute G and F addresses attributes of G.
$$MAP_{\lambda X.[\ldots\sigma_F(e)]}(...Iscan(I, l_I)) \rightarrow MAP_{\lambda X.[\ldots\sigma_F(e)]}(...Iscan(I, l_I \wedge F))$$
This rule expresses the classical two-steps processing related to spatial indices: *filtering* and *refinement*.

Projection, $\pi_{\vec{A}}(e)$. When projecting out one or several set attributes, one must carefully check that these attributes are not empty. In other words, the projected tuple must be valid. The test is useful only if some lazy operation has been performed previously.

1. **TRproj1** (The Fourier-Motzkin projection is always a candidate operation for set-attribute). If $Set(e)$:
$$\pi_{\vec{A}}(e) \rightarrow PROJ^{FM}(\vec{A}, e)$$

2. **TRproj2** (Check the validity of the tuple before projecting out set attributes).
   If $Relation(e) \wedge sch_{geom}(e) \not\subseteq sch(\vec{A}) \wedge Lazy(e)$,

$$\pi_{\vec{A}}(e) \rightarrow Norm \circ PROJ^{rel}(\vec{A}, e)$$

3. **TRproj3** (Simple relational projection when possible).
   If $relation(e) \wedge (sch_{geom}(e) \subseteq sch(\vec{A}) \vee \neg Lazy(e))$

$$\pi_{\vec{A}}(e) \rightarrow PROJ^{rel}(\vec{A}, e)$$

4. **TRproj4** (Specific algorithm for dimension 2). If $dim(e) = 2 \wedge Set(e)$, then

$$\pi_{\vec{A}}(e) \rightarrow PROJ^{dim2}(\vec{A}, e)$$

Join, $\bowtie$, Intersection $\cap$, Cross product $\times$. Those three operators are first uniformly translated in a generalized binary matching operation, $Join(p, e_1, e_2)$.

1. $e_1 \bowtie_p e_2 \rightarrow Join(p, e_1, e_2)$
2. $e_1 \cap e_2 \rightarrow Join(p_\cap, e_1, e_2)$, where $p_\cap$ stands as a conventional intersection predicate.
3. $e_1 \times e_2 \rightarrow Join(<>, e_1, e_2)$

Binary matching operation, $Join(p, e_1, e_2)$

1. **TRjoin1** (Use join predicates for inner relational index traversal whenever possible). If $Relation(e_1) \wedge Relation(e_2) \wedge e_2 = (...Iscan(I_{\vec{A}}, < l >))$ and any permutation of $sort(p')$ is a prefix of $\vec{A}$ for some $p' \subset p$
$$Join(p, e_1, e_2) \rightarrow Join(p'', e_1, ...Iscan(I_{\vec{A}}, < l \wedge p' >)))\ with\ p = p' \cup p''$$

2. **TRjoin2** (Sort-Merge algorithm if p relies on relational attributes and e_2 is not accessed through an index). If $Relation(e_1) \wedge Relation(e_2) \wedge sort(p) \subseteq sch_{rel}(e_1) \cup sch_{rel}(e_2) \wedge e_2 \neq (...Iscan...)$,

$$Join(p, e_1, e_2) \rightarrow Join^{rel}_{sm}(p, e_1, e_2)$$

3. **TRjoin3** (Nested loop if one cannot sort or if one index exists).
   If $Relation(e_1) \wedge Relation(e_2) \wedge (sort(p) \not\subseteq sch_{rel}(e_1) \cup sch_{rel}(e_2)) \vee e_2 = (...Iscan...)$

$$Join(p, e_1, e_2) \rightarrow Join^{mode}_{nl}(p, e_1, e_2)\ with\ mode \in \{lazy, norm\}$$

4. **TRjoin4** (Rule for set intersection).

$$MAP_{\lambda X.[...Join(p_\cap, X.i, X.j)...]}(...) \rightarrow MAP_{\lambda X.[...\cap^{mode}_{nl}(e_1, e_2)...]}(...)\ mode \in \{lazy, norm\}$$

5. **TRjoin5** (Use pointset bounding box for index traversal).

$$MAP_{\lambda X.[...Join(p, X.i, X.j)]}(\ldots Iscan(I_g, < l_I >)) \rightarrow$$
$$MAP_{\lambda X.[...Join(p, X.i, X.j)]}(\ldots Iscan(I_g, < l_I + bb(X.j) >)))$$

   $withX.j \in sch_{geom}(R_2)$. $bb(X.j)$ denotes the bounding box of $X.j$.

Difference, $e_1 - e_2$. Trivial: choose the algorithm according to the type of e_1 and e_2.

Union, $e_1 \cup e_2$ Trivial in every cases: just concatenate e_1 and e_2, possibly with duplicate removal.