DEDALE, A Spatial Constraint Database*

Stéphane Grumbach[1], Philippe Rigaux[2], Michel Scholl[1,2], and Luc Segoufin[1]

[1] INRIA, Rocquencourt BP 105, F-78153 Le Chesnay, France — {Stephane.Grumbach,Michel.Scholl,Luc.Segoufin}@inria.fr
[2] Cedric/CNAM, 292 rue St Martin, F-75141 Paris Cedex 03, France — {scholl,rigaux}@cnam.fr

Abstract. This paper presents a first prototype of a constraint database for spatial information, DEDALE. Implemented on top of the O_2 DBMS, data is stored in an object-oriented framework, with spatial data represented using linear constraints over a dense domain. The query language is the standard OQL, with special functions for constraint solving and geometric operations. A simple geographical application from the French Institute for Geography, IGN, is running on DEDALE. The data initially in vector mode was loaded into the database after a translation to constraint representation. Although it is too early to speak of performance since not all operations have been optimized yet, our experience with DEDALE demonstrates already the advantages of the constraint approach for spatial manipulation.

1 Introduction

General purpose database management systems (DBMS) are not well-suited for new applications that involve multidimensional data. This is due to the structural complexity of multidimensional data, which requires new querying facilities, and specific indexing techniques [LJF94,VG96], not provided by most existing DBMS.

The amount of spatial information is growing extremely rapidly. The manipulation of such information is one of the very important challenges of database technology [Gue94]. To satisfy the need, different systems have been developed for various subareas related to multidimensional data. Among them, Geographical Information Systems (GIS) constitute a large field of interest [ABC93,EH95].

Most systems devoted to the manipulation of geographic data are based on an architecture coupled with a relational DBMS, but dedicated to geographic information with special purpose tools for manipulating spatial data. This is the case of one of the most prominent systems for geographic information, namely ARC/INFO [Mor89]. Although these systems have been widely used for more than a decade, it is clear that the lack of integration between spatial data processing and standard data management is rather not desirable. Among various limitations, there is no high level query facility: In order to reach a high level of performance, the user has to be an expert in the system and use low level efficient primitives. There has been a large number of proposals in the literature [Gue94] for integrating spatial features with other traditional thematic alphanumeric features of geographic objects in a single framework. However, because of the complexity of spatial data and the variety of GIS applications, there has not been a common agreement on a satisfactory model for representing geographic data. Moreover for applications dealing with data of dimension higher than 2 (geology, spatio-temporal applications, etc.), there is a crucial lack of models and query languages.

GISs use two alternative modes for the representation of geometric objects. In the *raster* mode, data consists of an unstructured bitmap which renders the detection of geometric objects (lines, zones, etc.) difficult. The *vector* format is commonly

* Work supported in part by TMR project Chorochronos.

used, whether segment-based or point-based, to provide a structured representation of the geometric objects, in terms of polylines and polygons, defined by the list of their vertices or alternatively a list of segments.

The constraint database model introduced by Kanellakis, Kuper and Revesz [KKR90], turns out to be a good candidate for a sounder representation of spatial data. Linear constraint databases have been extensively studied recently, because of their potential application to geographical databases [VGV95]. One of the most promising features of constraint databases is their declarative query language based on first-order logic, and the existence of equivalent algebras, leading to a procedural expression of queries which offer potential for optimization. Indeed one of the drawbacks of the models proposed in the literature for representing and manipulating geographic information is that there is little framework for describing the semantics of the operations necessary for optimization.

The tight correspondence between the vector mode representation of GISs and the linear constraint framework is remarkable. There is an easy translation from one to the other. The main difference between the two representations is that in the linear constraint framework the objects are modeled as collections of simple convex polygons, while in the vector model, the polygons can be non-convex. There are efficient algorithms to perform polygon convexification. Moreover, most algorithms are both simpler and more efficient on convex polygons. The constraint representation simplifies the programmer's work, and might result in an increased efficiency.

While most GIS data models are not extensible, one of the nice features of the constraint model is the ability to handle more than 2 dimensions. This should be particularly useful for representing information in earth science applications.

Two families of algebras where defined for manipulating constraint relations: first, algebras that correspond exactly to the traditional relational algebra, and whose semantics is defined with respect to the effect of the usual algebraic operators on infinite relations [GST94,PVV94], and second, algebras that apply on special encoding for relations and generalized tuples [KG94]. We follow the first trend in the present approach.

The complexity of linear first-order queries was shown to be in NC^1 [GS97]. The expressive power of first-order logic with linear constraints has been broadly investigated [GST94,PVV95,GS97]. This language turns out to be rather restricted, in a way similar to the case of finite relations: For example, transitive closure and topological connectivity cannot be expressed in this language.

In this paper, we describe the first version of a prototype of a linear constraint database, DEDALE, and a simple geographical application. To the best of our knowledge, this is the first attempt to implement such a system. The C^3 system [BS97] differs strongly from DEDALE. It relies on a constraint calculus instead of an algebra, and the implementation is done on top of ObjectStore. Moreover, the application range as well as the prototypical queries are distinct.

The application is based on data from the French National Geographic Institute, IGN. It contains, for the Orange county in Provence [May90], some typical data as commonly found in GISs, such as ground occupancy, roads, powerlines, rivers, etc.

In the present setting, the data is modeled in an object-oriented framework. Essentially, spatial objects have an identifier, *thematic* attributes, and a spatial attribute expressed as a constraint relation. Therefore two limitations of the pure constraint model are overcomed. First spatial objects can be manipulated as first class citizens, by means of their identifier. In pure constraint models, the quantification is only over numbers, and not sets. The necessity to consider sets of points in the space explicitly has been addressed in particular in [GS95,PSV96]. The second restriction results from the rather limited expressive power of first-order linear queries in this context. Functions such as measures of distances for instance cannot be expressed since they are not linear. In DEDALE, they are expressed in the general purpose standard query language OQL [Cat94].

The paper is organized as follows. In the next section, we present the architecture of DEDALE and the current state of the implementation. The algebra for linear constraint relations is then presented in Section 3. In Section 4, we describe the geographical application, explain the data representation, and give examples of queries involving spatial operations. The main algorithms used in DEDALE for constraint solving, geometry, and query evaluation are described and analyzed in Section 5. The last section presents our ongoing work and perspectives.

2 Architecture of DEDALE

In this section, we present our architectural choices and the current state of implementation of the prototype. DEDALE has been implemented on top of the O_2 Database Management System (DBMS) [BDK92]. The architecture is forecast to deal with n-dimensional data. Present algorithms for implementing algebraic operations in the n-dimensional case are not yet optimal though.

To prove the feasibility of our approach, we have focused for the first version of the current prototype on 2-dimensional data and chosen a simple geographic application illustrating the features of DEDALE, described in Section 4. The implementation of the algebraic operations presented in the appendix is rather efficient. However in this first setting query processing is not yet fully optimized. Figure 1 illustrates the architecture of DEDALE. It consists of the following components:

1. A Graphical User Interface (GUI).
2. The DEDALE processor composed of the following modules: (i) the lexical and syntactical analyzer of an algebraic query which requires the access to the data dictionary stored in the O_2 database, (ii) the query optimizer which transforms an algebraic expression into an optimal execution plan. It relies on the existence of indices for a fast data access. It generates OQL queries calling functions of (iii) the *constraint solving engine.*
3. The O_2 DBMS providing OQL for querying the database.

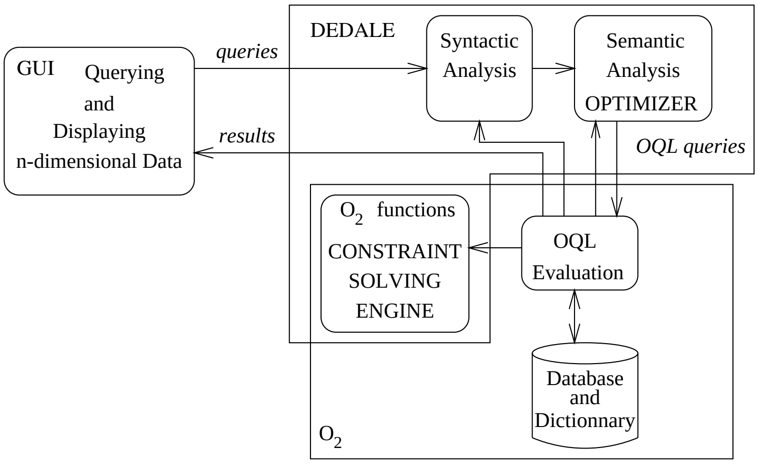


Fig. 1. Architecture of DEDALE

The GUI allows to query the database with query parameters such as points or rectangles, and to display query results including maps and other thematic data. The standard query language OQL is used for querying the database. The spatial data is dealt with through a constraint solving engine. It has been designed as a set of constraint solving operations, implemented as O_2 functions written in C. Since O_2 functions in the above set can be called in an OQL query, this design allows to mix constraint solving operations with classical query evaluation. However, the problem of query optimization is only partially addressed in the current prototype. Only spatial data has been implemented by constraints and O_2 functions, implementing algebraic operations on constraint relations representing convex objects in the 2D plane, are used in the body of OQL queries.

We might have chosen to implement descriptive attributes by (single equality) constraints as well, and chosen, as a query language on these relations, the operations of the algebra as defined below. Instead, for simplicity in this first implementation, we chose to rely on the existence of the O_2 data model (and the query language OQL) for the representation (and query) of thematic attributes, restricting the use of the constraint model and associated algebra to the representation of the geometry and for solving spatial predicates and operators.

3 Constraint algebra

We present in this section the algebra for linear constraint databases, which it equivalent to first-order logic. We consider the classical algebra introduced by Codd for finite relations [Cod70], in the case of infinite relations, that is that the operators apply to any relation, finite or infinite. We define a *symbolic algebra*, with operators applying to the finite representations of the possibly infinite sets, and whose effects on the finite representation corresponds exactly to the operations on the infinite relations.

The symbolic algebra consists of the following operations: Cartesian product, $\times$, projection, π, union, $\cup$, set difference, $-$, and rename, ρ. The algebra operations are performed on sets of generalized tuples, i.e. on quantifier-free formulas in *disjunctive normal form*.

Suppose R is an n-ary relation represented by a quantifier-free formula, φ, of the form:

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

where the $\varphi_{i,j}$'s are atomic formulas (simple linear constraints with integer coefficients using equality or inequalities predicates). Then, we also denote the representation φ as a collection of generalized tuples t_i in the set notation:

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

We now describe the effect of symbolic operators on sets of generalized tuples. Let R_1 and R_2 be two relations, and respectively e_1 and e_2 be sets of generalized tuples defining them.

1. $R_1 \times R_2 = \{t_1 \wedge t_2 \mid t_1 \in e_1, t_2 \in e_2\}$.
2. $\pi_{\overline{x}}\, R_1 = \{\pi_{\overline{x}}\, t \mid t \in e_1\}$,
   where

$$\pi_{\overline{x}}\, t = \bigwedge_{1 \leqslant k \leqslant K, 1 \leqslant \ell \leqslant L} \quad b^k\overline{x} - b_0^k \leqslant a_0^\ell - a^\ell\overline{x} \;\wedge \bigwedge_{1 \leqslant i \leqslant I} \quad c^i\overline{x} \leqslant c_0^i.$$

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 normalized):

$$\begin{cases} a^{\ell}\overline{x} + y \leqslant a_0^{\ell} & \text{for } \ell = 1, \ldots, L \\ b^{k}\overline{x} - y \leqslant b_0^{k} & \text{for } k = 1, \ldots, K \\ c^{i}\overline{x} \leqslant c_0^{i} & \text{for } i = 1, \ldots, I \end{cases}$$

where $\overline{x}$ ranges over $\mathbb{Q}^n$, and y over $\mathbb{Q}$.

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

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

The behavior of symbolic operators implies computation at different levels of the formulae. The representation in DNF has various imbricated structural levels: the disjunction, the conjunctions, the predicates of the atomic constraints, and the parameters of the constraints. Union has an effect only at the upper level of the disjunction. Cartesian product has an effect at the level of the intersections. Set difference has an effect at all levels till the predicates of the constraints, which can be inversed. Finally, projection has an effect at all levels including the parameters, and implies some numerical computation.

It was shown in [GST94] that the algebra $\mathrm{ALG}_{\mathcal{L}}$ 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.

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).

Section 5.3 describes the current implementation in DEDALE of the algebraic operators. Each operator checks its input's bounding boxes first and, if needed, applies different algorithms depending on the dimension. For example, in dimension 2, projection is easily obtained by projecting the bounding box on the corresponding axis, negation uses techniques from cell decomposition, etc. (See Section 5.3 for more details). All algebraic operations correspond to pure symbolic constraint manipulation. Only an operation called *normalization*, which puts tuples in a normal form described in the following section, actually evaluates the query, i.e. checks for tuple or relation satisfiability. Normalization can be postponed until the end of the execution plan, which is not necessarily optimal. Choosing in the execution plan when to execute normalization remains a research issue. A worst case analysis has been studied in [GL95].

4 A geographic application implemented with DEDALE

In this section, we present the application that is running on DEDALE. We first consider the geographic objects, and then give a sample of queries, using successively the algebra and its implementation with OQL and the constraint solving operations.

The experiment uses a dataset provided by the French *Institut Géographique National* (IGN) and extracted from their "Base de données topographique" [BDT91]. Geographic entities are samples of features commonly found in GIS: *Ground occupancy* (denoted *GO* in the following), *Roads*, *Powerlines*, *Rivers*, *Water Points*, *Buildings* and *Tracks*.

In conventional 2-dimensional Geographic Information Systems (GIS), each geographic entity is represented by *descriptive* alphanumeric attributes (thematic attributes), such as the ground type and the owner's name for ground occupancy, and by a geometric attribute representing the location on earth as well as the geometry of the entity. This attribute usually is in a *vector* format.

It is noteworthy that most of the time, the geometry is not convex. A road, for instance, is spatially represented as a list of consecutive segments (a *polyline*). Polygons usually are *simple* but not convex.

Each geometric object must then be split into convex components in order to satisfy the constraint model requirement that each tuple be a conjunction of linear constraints: therefore each geographic object is associated with a set of generalized tuples with the same descriptive attribute values.

We chose the following O_2 schema for implementing geographic objects. (For illustration we only give a schema for *GO*. A similar schema can be given for *Roads*, etc). Each geographic object is represented by some descriptive attributes followed by a geometric attribute of type *Relation* (a generalized relation) representing the set of its convex components: each generalized tuple has two attributes (x and y coordinates in the plane).

class *GO*
tuple [*No:* **integer**,
    ground-type: **string**,
    owner: **string**,
    geometry : Relation]

The sample database has the following characteristics: the main O_2 file is 30MB. The number of instances for each entity ranges from 20 (*waterpoints*) to 600 (*buildings*). The *GO* collection contains 307 parcels which represent the most complex spatial objects: the average parcel is a simple polygon with 80 vertices, some parcels having up to 300 vertices.

4.1 Constraint representation of spatial data

Before giving some examples of queries, let us look at the representation of constraints in O_2. Generalized relations are implemented according to the following schema:

type *Relation*
tuple [*mbb: Rectangle,*
    tuples : **set** *(Tuple)*]

type *Tuple*
tuple [*mbb: Rectangle,*
    constraints : **list** *(Constraint),*
    geo-type : **GEO-TYPE**
    normal : **Boolean**]

where **GEO-TYPE** = $\{Point, Segment, Polygon, Unknown\}$, *Constraint* is defined below.

The schema is a straightforward implementation of the generalized relational model, as presented in Section 3. The following information, mostly intended to speed up the evaluation process, have been added to the schema:

1. A first step towards optimization is achieved by associating with each tuple its *minimal bounding box* (*mbb*) defined as the tuple ((min$_1$, min$_2$,..., min$_n$), (max$_1$, max$_2$, ..., max$_n$)) of points in the n-dimensional space with minimal (maximal) coordinates satisfying the constraints. Using *mbb*'s in query evaluation is very common: it allows a first fast filtering step. The relation's *mbb* is also added. It is defined as the *mbb* of the *mbb*'s of its tuples.
2. The *normal* attribute is a **Boolean** set to "true" if the tuple is in *normal form*, defined as follows:
   (a) There does not exist any couple of redundant constraints in the list *constraints*. For instance, $x < 5$ and $x < 1$ cannot belong to the same normalized tuple.
   (b) There does not exist any couple of inconsistent constraints in *constraints*. For instance, $x > 5$ and $x < 1$ cannot belong to the same normalized tuple. It follows that a normalized non-empty tuple is always satisfiable.
   (c) Although not central to the normalization, the **list** constructor implies that the constraints are in a given order, important for displaying polygons: graphical devices rely on a point based representation of polygons (an ordered sequence of polygon vertices). The conversion constraints to point representation is of course faster (linear in the number of polygon vertices) if the constraints are in the right order.
3. While constraints are not typed in our data model, typing might be useful in the context of geographical databases. In particular in 2-dimensional geographical databases, a tuple is either a point, a polyline, or a polygon. In our prototype, this typing information will only be used for display purposes: data has to be displayed using graphical devices which make the usual distinction between points, lines and polygons.

Attribute *geo-type* gives the geometric type of the object to be represented by the tuple. Since a conjunction of constraints defines a convex object, this type can be either a *Point*, a line *Segment* or a convex *Polygon*. An *mbb* can be defined on any segment or polygon.

Unbounded objects (for instance a half straight line, or an infinite triangle defined by two intersecting half-planes) do not have any *mbb*. They are assigned the *Unknown* type. The *mbb* attribute is not relevant in this case.

It remains to define the type *Constraint*. Any constraint is of the form:

$$a_1x_1 + a_2x_2 \ldots + a_kx_k \; op \; b$$

where k is the arity (or dimension) of the relation, x_i is the variable associated with the A_i attribute, each a_i and b are rational and *op* belongs to **OP** $\{=, <, \leqslant, \geqslant, >\}$. Therefore a constraint is implemented as

type *Constraint*
tuple [*constants:* **list** (rational)
    op : **OP**]

where the vector of constraints *constants* is a list of $k+1$ rationals.

This concludes the logical schema related to constraints representation. However, in the current implementation, a more physical implementation has been used: instead of using the **list** constructor for storing the constraints of a tuple, we consider a constraint as a binary object and represent the tuple's constraints as an array of bytes. This allows an efficient clustering of constraints which are always to be read together, and avoids the penalty of indirections in the O_2 **list** constructor.

4.2 Mapping non-convex objects to a constraint-based representation

Mapping non-convex objects to a constraint-based representation (a generalized relation) is necessary at least for data loading since data in the outside world is available in the vector format. The conversion is trivial for points, which are represented by two constraints of the form $x = a \wedge y = b$. Things are a bit more complex for polylines and polygons.

Polylines: A polyline is an ordered collection of connected segments whose point-based representation is a list of points $< P_1, P_2, \ldots, P_n >$. A segment P_iP_{i+1} can be represented by a tuple of the form $x_{min} \leqslant x \leqslant x_{max} \wedge y_{min} \leqslant y \leqslant y_{max} \wedge a_1x + a_2y + b = 0$, a conjunction of 5 constraints: one for the equation of the line collinear to the segment, and 4 for the bounding box.[1]

Polygons: A point-based representation of a polygon is also a list of points $< P_1, P_2, \ldots, P_n >$ describing the boundary as a polyline, adding a last segment P_nP_1 since the boundary is closed. If a polygon is not convex, it must be partitioned into convex components. This is done in two steps: first the simple polygon is triangulated in $n - 2$ triangles, second we merge adjacent triangles whose union is still convex in order to obtain a smaller number of convex components.
A tuple is created for each resulting convex polygon: the boundary is scanned in counterclockwise order, and for each edge e the constraint $a_1x + a_2y \leqslant b$ representing the half-plane located to the side of e is appended to the *constraints* attribute of the tuple.

It is worth noting that each tuple thereby inserted in the database is normalized: there are no redundant nor inconsistent constraint. In addition, the order of the constraints in a *Polygon* tuple is also the order of the segments along the boundary of the convex polygon. Thanks to this order obtained through the **list** constructor in the *Tuple* type, the vector representation can be computed back by a simple scan.

4.3 Examples of queries

To illustrate the power of the constraint approach and the functionalities of the current prototype, a sample of queries is given below, expressed in OQL as implemented in the prototype on objects whose schema was given in the previous subsection. The implementation of the constraint solving functions used is presented in Section 5. Formally, they rely on the algebra presented in the previous section. The entry points to the O_2 database (O_2 **names**) used are *GO, Roads, Waterpoints, Cities* and *Buildings*.

– Display the part of roads which is inside the rectangle @*rectangle* drawn on the screen.

  select *SELECT(r.geometry, @rectangle)*
  from *r* **in** *Roads*
  where *SATISFY (SELECT(r.geometry, @rectangle))*

  This query computes, for those roads that overlap the rectangle, the part inside the rectangle (*SELECT* call in clause **select**). *SELECT* (which implements the algebraic selection using the symbolic operator cross product) takes as inputs the set of convex tuples (*r.geometry*) representing the geometry of r and the @*rectangle* constraints. *SATISFY* is true if the result of *SELECT* is non empty.

[1] This is not optimal since a segment can be described by 3 constraints. However the bounding box is already stored as part of the tuple.

– *No* of the forests crossed by a road.

```
select   o.No
from     o in GO, r in Roads
where    o.ground-type = "forest"
and      SATISFY (INTERSECTION(o.geometry, r.geometry))
```

The join between a forest geometry and a road geometry is performed by the *INTERSECTION* function (implementing algebraic operator *INTERSECTION*). *SATISFY* is true if the result of *INTERSECTION* is non empty.

– *No* of the forests crossed by a road inside the rectangle @*rectangle*?

```
select   o.No
from     o in GO, r in Roads
where    SATISFY (SELECT(INTERSECTION(o.geometry,
                                      r.geometry),
                         @rectangle))
and      o.ground-type = "forest"
```

This query consists of a join between *Roads* and *GO*, which yields forests crossed by roads, and a selection which keeps that part of the previous result inside the user-defined rectangle.
Note that the order of operations in the OQL expression, intersection first, followed by selection, is not necessarily optimal.

– Waterpoints south of Orange.

```
select   w.No
from     w in WaterPoints, c in Cities
where    c.City-name="Orange"
and      SATISFY(SELECT(JOIN(JOIN(w,PROJECT(c.geometry,x)),
                             RENAME(PROJECT(c.geometry,y),
                                    y→y')),
                        y < y'))
```

The inner join keeps the waterpoints whose abscissa is inside "Orange" abscissas range. The second join keeps those whose ordinate is smaller than the ordinate of at least one point in "Orange".
Finally the following query uses the algebraic difference as defined in Section 3 and denoted *DIFF*.

– Buildings surrounded by a forest.
In the OQL expression below, the "**where** *SATISFY*" clause selects the buildings included in and intersecting forests, while the "**and** ¬" clause selects the buildings intersecting or outside forests.

```
select   b
from     b in Buildings, f in GO
where    SATISFY (INTERSECTION(f.geometry,b.geometry))
and      ¬SATISFY (INTERSECTION(NOT(f.geometry),(b.geometry)))
and      f.ground-type = "forest"
```

All the queries commonly found in spatial database applications (*clipping*, *windowing*, *point-in-polygon*, *map overlay*, etc.), as well as some less common requests can be expressed in DEDALE using the small set of operations above.

Figure 2 illustrates the GUI of DEDALE. The top-left window allows to enter OQL queries as shown above. The results obtained during a querying session can be displayed in several maps, each showing a subset of the whole data retrieved from the database [PR95]. MAP0 in Figure 2 contains several kinds of geographical objects (ground occupancy, roads, waterpoints). MAP1 displays a subset of MAP0: only objects of *GO* are selected, and only the part of the parcels included in a given rectangle referred to by \$1. MAP1 is the result of a query expressed as explained previously.

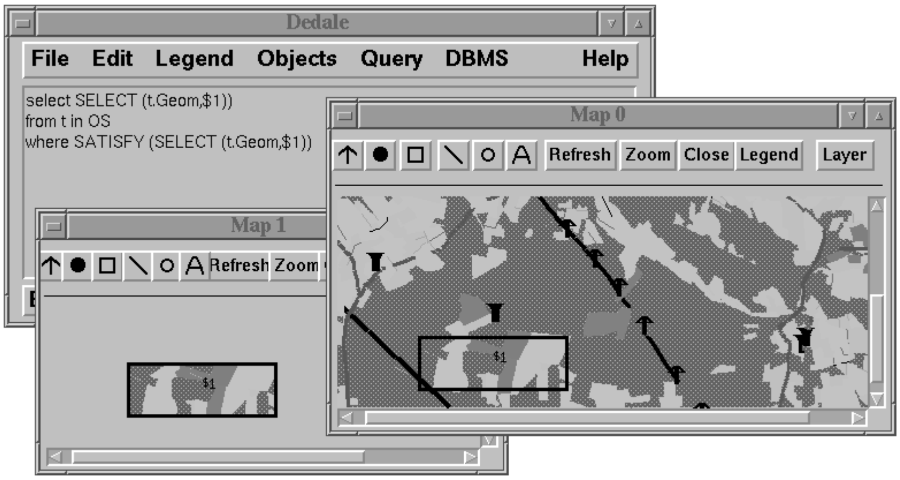


Fig. 2. Interactive querying with DEDALE

5 Constraint solving and geometric algorithms

In this section, we describe the current implementation of DEDALE. The presentation is bottom-up: we begin with *geometric operations* that are used in *constraint manipulation primitives*, which in turn are the basis to evaluate the *algebraic operators* of Section 3. The section ends up with a short discussion on the query evaluation strategy currently implemented in DEDALE.

5.1 Geometric operations

The first goal of these operations is to interface DEDALE with the outside world, both for *data loading* since geographical data is usually provided in vector mode, and for *data display* using graphical softwares which rely on a vector representation.

Data loading (function *ToConstraint*)

The function *ToConstraint* transforms a point-based representation towards its equivalent constraint representation. The conversion is trivial for points and simple for lines (see Subsection 4.2) but deserves some further explanations for polygons.

Many algorithms have been designed for polygon partition, mostly for polygon triangulation [GJPT78,FM84,TvW88,Cha91,Sei91]. However, triangulation results in a large possibly prohibitive number of convex components in the partition. Given a polygon with n vertices, the number of triangles in the partition is $n-2$. Thus, we would like to minimize the number of convex elements in the partition. There exists an algorithm proposed in [Kei85] which gives the optimal number of convex components. Unfortunately, in addition to being rather complex, the algorithm time complexity is large: $O(r^2 n \log n)$ where r is the number of "notches".[2] Hence, we preferred to implement an algorithm suboptimal in terms of the number of components, but simple to implement and running in linear time [HM83].

The algorithm takes as input a set of triangles obtained by triangulation,[3] together with the adjacency graph produced by the former triangulation algorithm, a binary tree representing adjacent triangles. This tree is scanned bottom up: one first tests whether the union of triangle t_1 with another adjacent triangle t_2 is convex. If yes one keeps scanning the tree up and merging a convex polygon with adjacent triangles until the object is no longer convex. The process is then repeated with another leaf if any.

The algorithm is linear in the number of vertices of the initial polygon. As a result, we get a set of convex polygons whose number of elements, as shown in [HM83], is never worse than four times the optimal.

Table 1 gives some statistics on the convex partitioning of the polygons of the *GO* collection. There are 307 simple polygons, with an average of 80.87 vertices per polygon. The partitioning yields 10 457 convex components with an average number of vertices of 4.46. The average number of "notches" is a measure of the "non-convexity" of the polygon: 41.23% of the vertices are notches.

Avg(vertices) (SP)	% of notches	Avg(convex/notches)	Avg(vertices) (CC)
80	41.23	0.95	4.46

Table 1: Convex partitioning statistics[4]

If r is the number of notches, the optimal number of components is in $[\lceil \frac{r}{2} \rceil + 1, r+1]$ [CD85]. Table 1 shows that we get almost r (0.95) in spite of the algorithm sub-optimality. This proves the efficiency of our implementation, but the large ratio (approximately 30) between the number of convex polygons and the number of simple polygons illustrates the need for an efficient representation of constraints (generalized tuples).

Data display (function *ToVector*)

The function *ToVector* takes as input a *normalized* tuple and delivers the corresponding vector representation. As shown in Subsection 4.2, the design of a normalized tuple makes this operation quite easy.

Half-Plane Intersection

The third geometric operation, *Half-Plane Intersection*, denoted HPI, optimizes 2-dimensional constraint solving. Given a set of half-planes, this operation computes the associated convex polygon. We implemented the algorithm of [PS85] which is optimal and whose time complexity is $O(n \log n)$ where n denotes the number of half-planes.

5.2 Constraint manipulation

Tuple *normalization* is central to the implementation of the algebra. Normalization eliminates redundancy. If the resulting set is empty, then the tuple is unsatisfiable. The normalization algorithm works as follows: first the constraints are scanned in order to count the number of constraints with operator *equal*. Except for degenerated

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

[3] The triangulation algorithm implemented here is the "greedy" $O(n^2)$ algorithm described in [O'R94].

[4] SP stands for simple polygons, and CC for convex components.

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* computes a convex polygon which is converted into constraints.
The algorithm is given below[5]: CEQ denotes the subset of $t.constraints$ such that op is '=' (straight lines), CNE is $t.constraints - CEQ$ (half-planes). The MBB function computes the bounding box of any point-based geometric object.

Function *NORMALIZE*
Input: *a tuple t*
Output: *a normalized tuple* t'
begin
  $t'.constraints := \emptyset$; $t'.type := Unknown$; $t'.mbb := \emptyset$; $t'.normal := true$
  if $Card(CEQ) > 2$ **return** t' // *Unsatisfiable*
  else if $Card(CEQ) = 2$ **do** // *A point pt*
  begin
    Check that each constraint in CNE contains pt
    if *yes* **then**
      t'.type := Point
      t'.constraints $\leftarrow (x = pt.x \wedge y = pt.y)$
      t'.mbb := MBB (pt)
    else *return* t'
  endif
  else if $Card(CEQ) = 1$ **then** // *A segment*
  begin
    t'.type := Segment; t'.mbb := MBB(HPI(CNE))
    t'.constraints $\leftarrow CEQ + ToConstraints(MBB(HPI(CNE)))$
  endif
  else if $Card(CEQ) = 0$ **then** // *A convex polygon*
  begin
    t'.type := Polygon; t'.mbb := MBB(HPI(CNE))
    t'.constraint := ToConstraint(HPI (t.constraints))
  endif
return t'
end

At the relation level, two other functions are useful for constraint solving. The first one *SATISFY*, Boolean, is set to true if the generalized relation argument is non empty. *SATISFY* simply scans the tuples of the relation, normalizes each tuple and returns "true" as soon as it finds a satisfiable tuple. The second function, *OVERLAP* takes as input two relations, R_1 and R_2, and performs a nested loop over both collections of tuples, taking the conjunction of each pair of tuples. A simple optimization is possible in order to eliminate some empty tuples: two tuples are concatenated only if the intersection of their respective bounding boxes is non empty. This algorithm works in the general n-dimensional case, and is currently implemented for 2-dimensional data.

Function *OVERLAP*
Input: *two relations,* r_1 *and* r_2
Output: *a relation* r_3
begin
  if $r_1.mbb \cap r_2.mbb \neq \emptyset$ **then**
  begin
    for each t_1 *in* r_1 **do**

[5] Note that to simplify its description, we assume that the input does not contain any degenerated case such as two constraints representing parallel straight lines, or unbounded objects.

```
        begin
            for each t2 in r2 do
            begin
                if t1.mbb ∩ t2.mbb ≠ ∅ then
                begin
                  Create a new tuple t
                  t.normal := false; t.type := Unknown; t.mbb := t1.mbb ∩ t2.mbb
                  t.constraints := t1.constraints ∧t2.constraints
                  Insert t in r3.tuples; r3.mbb := R3.mbb ∩ t3.mbb
                endif
            enddo
        enddo
    endif
return r3
end
```

Note that the tuples of the resulting relation are not normalized: some constraints in a tuple might be redundant or inconsistent.

5.3 Implementation of the algebraic operators

We now describe the current implementation of the algebraic operators. Recall that the constraint-based representation is restricted to generalized relations of arity 2 for geometric data. At the physical level, we consider various operations not distinguished at the logical level. This is the case of join, intersection, and selection, which can all be expressed at the logical level by a Cartesian product.

Projection, π

$r = \pi_{\bar{x}}(r_1)$ is defined as the relation whose tuples are the projection of each tuple of r_1 on the $\bar{x}$ axis. To compute the projection of a tuple in the general case, we use an algorithm based on the Fourier-Motzkin method.

However, when restricted to dimension 2 there is an easiest and fastest way to perform projection. First each tuple is normalized; during normalization, the tuple minimal bounding box (*mbb*) is computed; then the *mbb* is projected on the correct axis: this gives the result.

Join, $\bowtie$

The join $r = r_1 \bowtie r_2$ is obtained by computing the conjunction of each tuple of r_1 with each tuple of r_2. The conjunction of two tuples is just the concatenation of their constraints. The resulting set is *not* normalized, i.e. checked for satisfiability and redundancies. In the 2-dimensional case, the join is implemented by a direct call to the above *OVERLAP* function.

Intersection, $\cap$

$r = r_1 \cap r_2$ is a special case of join. Here r_1 and r_2 are supposed to be defined on the same attributes. In the current implementation of DEDALE (2-dimensional data), the operator is implemented by a direct call to the *OVERLAP* primitive.

Union, $\cup$

$r = r_1 \cup r_2$ is defined as the set of tuples which belong either to r_1 or to r_2 and is simply implemented as the concatenation of the tuples lists from r_1 and r_2.

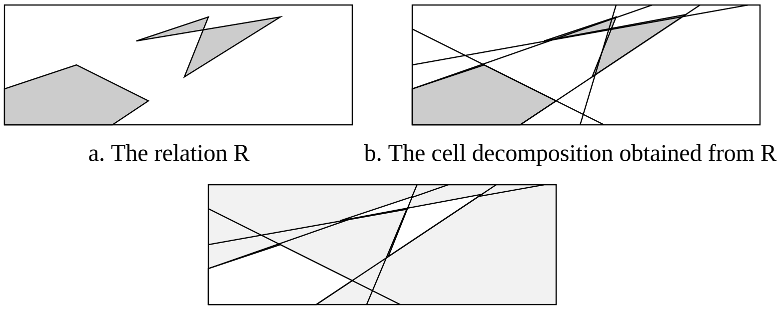
Selection, σ

$r = \sigma_F(r_1)$ is easy to implement since F is just a constraint which has to be added to each tuple of r_1. For 2-dimensional data, selection uses once more the

OVERLAP(r_1, r_2) primitive, r_2 being in that case the F singleton. Once again, r is not normalized: it might contain empty tuples.

Difference, $-$

$r = r_1 - r_2$ gives the set of points that are in r_1 but not in r_2. $r_2 = \bigvee \bigwedge c^1_{ij}$, therefore its negation is $\neg r_2 = \bigwedge \bigvee \neg c^1_{ij}$ which is a conjunctive normal form (CNF) formula. As $r = r_1 \cap \neg r_2$, computing r is transforming a CNF formula into a DNF one. This transformation is exponential in general. However when the dimension (arity) is fixed it is possible to compute r in polynomial time. For dimension 2, which is our case of interest here, it is possible to compute negation in time $O(n^3)$ where n is the number of constraints of r_1 and r_2. This algorithm is maybe not the best, but the technique of *cell decomposition* is interesting by itself and could be reused to compute the connectivity of a region for example. The idea is roughly to get a *cell decomposition* of the plane using the constraints of r_1 and r_2. Each constraint of r_1 or r_2 gives a line in the plane. All the lines partition the plane into *cells*. One can show that r_1 and r_2 are unions of such *cells* and therefore r is the union of the *cells* that are part of r_1 but not part of r_2. See Figure 3. The computation of the cells, together with the constraints defining it, can be done in time $O(n^2 \log n)$ where n is the number of constraints in r_1 and r_2. Then we just have to check whether a *cell* c belongs or not to a relation r_i. This can be done in time linear on the size of r_i ($O(n)$) because it is enough to check whether a point of c is in r_i. As there might be n^2 such cells, the overall computation time is $O(n^3)$.



a. The relation R

b. The cell decomposition obtained from R

c. The negation of R

Fig. 3. negation

Function *DIFF*
Input: *two relations,* r_1 *and* r_2
Output: *a relation* r
begin
    First get the list of all the line equations in all constraints in all tuples of r_1 *and* r_2.
    Those lines partition the plane into cells *which are convex areas. See fig 3 step 2.*
    Compute the constraints defining the cells.
    for each cell c **do**
    begin
        if $c \subset r_1 \wedge c \not\subset r_2$ **then**
        begin
            Create a new tuple t
            $t.normal := true;\ t.type := Area;\ t.mbb := c.mbb$
            $t.constraints := c.constraints$
            Insert t *in* $r.tuples;\ r.mbb := r.mbb \cup c.mbb$

```
        endif
    enddo
end
```

Computing the cell decomposition is the hardest part. This can be done in time $O(n^2 \log n)$ where n is the number of lines involved using the following algorithm.

Assume that we are given a list L of lines in the plane. Let L_p be the list of all their intersecting points. A cell is then completely defined by any of its vertices with the corresponding *angular sector*. If all lines are sorted on their slope, all *angular sectors* of a given point p are defined by two successive lines intersecting at p. To compute the cell from this point we need to be able to go from one point of L_p to the next one in the same line. This can be easily done by sorting for all lines the intersection points with the other lines using their x, y coordinates. All cells have been computed when all *angular sectors* of each point have been used. During the computation of L_p it is possible to compute the list of *angular sectors* for each point. Each time a point is "used" for computing a cell we mark it as used. When they are all used, the corresponding point is removed from L_p. The 2 last operations are done by a function *visit(p)*. When L_p is empty, all cells have been computed.

Function *Linear Cell Decomposition*
Input: *a list of lines equations L*
Output: *a relation r which generalized tuples are precisely the cells*

```
begin
    1) Sort lines using their slopes.
    2) Compute the intersecting points, together with their angular sectors.
    3) Sort the points in L_p
    NewRelation(r) // init a new relation
    while L_p ≠ () do
        NewTuple(t) // init a new tuple
        (p,o) ∈ L_p // get a point which has not been visited enough time,
        line_beg = first(p,o) // get the corresponding line equationss,
        line_cur = second(p,o) // which are used to define the first constraints for one the cell.
        Visit(p,o); record that (p,o) was visited, remove p of L_p if needed.
        AddConstraint(t, line_beg);
        AddConstraint(t, line_cur);
        while line_cur ≠ line_beg do
            get the next point in the current line.
            // This is possible in constant time using the index computed at step 2)
            (p,o) = Next(p,o);
            line_cur = second(p,o);return the next line in order after line_cur in (p,o).
            AddConstraint(t, line_cur) // which is the one that will define
            Visit(p,o);
        enddo
        AddTuple(r,t);
    enddo
end
```

Function *NOT*

The complement operation, denoted $\neg$, is implemented by the function NOT. $r = \neg r_1 = \mathbf{R}^2 - r_1$ is a variant of difference: the cell decomposition is done using only the lines of the constraints of r_1. Then one just has to check whether a given cell is or not part of r_1.

5.4 Query evaluation

As shown previously, all the operators can be done by only symbolic manipulation: $\cup$ manipulates relations, $\times$ manipulate tuples, but none of them checks for tuple's or relation satisfiability. Hence, the actual evaluation of the query is delayed until a call to the *NORMALIZE* operation.

This suggests that choosing the right place in the execution plan to carry out the normalization process is an essential part of the query processing strategy. Such strategies were proposed in [GL95]. The current implementation of DEDALE uses the simplest one, *lazy evaluation*. Normalization is only performed once (except for projection), as the root operation of the query execution tree. This *lazy evaluation* was shown to be rather inefficient. Needless to say, we plan to experiment some more sophisticated evaluation strategies, and integrate them in the optimizer module.

6 Perspectives

The main challenge of such an implementation was to prove the feasibility of a constraint database approach for representing and querying geographic information. The theoretical advantages of the approach, widely addressed now in the literature are a sound mathematical foundation for spatial data representation which is extensible and the possibility of defining query languages based on the classical relational framework. In practice, the linear constraint model corresponds nicely to the vector mode representation of GIS in the sense that the same 2-dimensional data can be stored in both paradigms. The promising features are:

– A simple powerful model applicable to a large variety of geographical applications, with no limitation on data dimension.
– A high level declarative language for querying geographical information.
– The potential for optimization given by the algebra. We are currently integrating spatial indices already implemented with O_2 [SGR96].
– Simple expression of queries that are cumbersome to express and process with current GIS, such as interpolation of values, interdependent values, queries involving direction predicates (North-West of, etc.).
– A potential for parallel evaluation. The use of constraints clarifies indeed what operations can be parallelized. Moreover, parallel complexity results for some subclasses of the full algebra lead directly to parallel algorithms.

The constraint solving engine has been completely implemented. It could have been possible to reuse partly existing softwares for constraint solving from linear programming for instance. Because of the specificity of our needs, we started from scratch and developed much simpler algorithms and code. For the 2-dimensional case the implemented functions are rather efficient. But one of the advantages of the constraint modelization is that it is not restricted to bidimensional data. It is easy to extend the current version of DEDALE to handle higher dimensional data in a fully modular way. All the algorithms proposed in Section 5.3 are valid in the n-dimensional case, as long as there exists an algorithm for normalization. Clearly the implementation of some of these operators in the n-dimensional case deserves more efficient algorithms. Mapping n-dimensional data onto 2-dimensional output devices also deserves a specific study.

Thus our first objective is to test DEDALE against an application handling data with higher dimensions. We plan to investigate a spatio-temporal application involving moving objects, where both time and space are modeled with constraints.

Our second objective is a full optimization of queries expressed on constraints (module *optimizer* in Figure 1). It includes as a specific task the design of indexing for spatial data represented as constraints. But the main problem here is to translate general algebraic queries into an optimal execution plan. The current implementation handles thematic attributes (with OQL) separately from spatial data (O_2 functions for solving constraints). This leads only to partial optimization through OQL query evaluation which does not take into account the O_2 function semantics for the optimization. We plan to implement an independent query parser and optimizer fully devoted to an efficient evaluation of constraint-based queries. Note that, for efficiency reasons, the query evaluator is aware of the dimension of the objects manipulated and calls distinct algorithms for dimension 1, 2 or more.

Acknowledgments

The authors wish to warmly thank Zoé Lacroix and Gabi Kuper for the numerous discussions that oriented the choices made in the present prototype. We are also indebted to Stéphane Solé who implemented the intersection operator and to Guillaume Masson who provided the convexification statistics. Thanks also to Manolis Koubarakis for comments on a draft of the paper, and to Guido Moerkotte for very helpful discussions on the algebraic operators.

References

[ABC93] D. Abel and B.-C.Ooi, editors. *Proc. Intl. Symp. on Large Spatial Databases (SSD).* LNCS No. 692. Springer-Verlag, 1993.

[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.

[BDT91] La Base de Données Topographiques de l'I.G.N. Bulletin d'Information de l'I.G.N., No. 59, 1991.

[BS97] A. Brodsky and V.E. Segal. The c^3 constraint object-oriented database system: an overview. In *Constraint Databases and Applications, Proc. second international workshop on Constraint Databases Systems (CDB97)*, Lecture Notes in Computer Science, pages 134–159, 1997.

[Cat94] R.G.G. Cattell, editor. *The Object Database Standard ODMG-93.* Morgan Kaufmann, San Francisco, California, 1994.

[CD85] B. Chazelle and D.P. Dobkin. Optimal Convex Decomposition. In G.T. Toussaint, editor, *Computational Geometry*, pages 63–133. North Holland, 1985.

[Cha91] B. Chazelle. Triangulating a Simple Polygon in Linear Time. *Discrete and Computational Geometry*, 6:485–524, 1991.

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

[EH95] M.J. Egenhofer and J. Herring, editors. *Proc. Intl. Symp. on Large Spatial Databases (SSD).* LNCS No. 951. Springer-Verlag, 1995.

[FM84] A. Fournier and D.Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Transactions on Graphics*, 3:153–174, 1984.

[GJPT78] M.R. Garey, D.S. Johnson, F.P. Preparata, and R.E. Tarjan. Triangulating a Simple Polygon. *Information Processing Letter*, 7(4):175–180, 1978.

[GL95] S. Grumbach and Z. Lacroix. Computing queries on linear constraint databases. In *Fifth Int. Workshop on Database Programming Languages*, 1995.

[GS95] S. Grumbach and J. Su. Dense order constraint databases. In *14th ACM Symp. on Principles of Database Systems*, pages 66–77, San Jose, May 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.

[Gue94] R. Gueting. An Introduction to Spatial Database Systems. *The VLDB Journal*, 3(4), 1994.

[HM83] S. Hertel and K. Mehlorn. Fast Triangulation of Simple Polygons. In *Proc. Intl. Conf. on Foundations of Computer Theory*, LNCS No. 158, pages 207–218. Springer-Verlag, 1983.

[Kei85] J.M. Keil. Decomposing a Polygon into Simpler Components. *SIAM Journal of Computing*, 14(4):799–817, 1985.

[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.

[LJF94] K. Lin, H.V. Jagadish, and Christos Faloutsos. The tv-tree - an index for high-dimensional data. *VLDB Journal*, 1994.

[May90] Peter Mayle. *A Year in Provence*. Knopf, New York, 1990.

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

[O'R94] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

[PR95] J.-P. Peloux and P. Rigaux. A Loosely Coupled Interface to an Object-Oriented Geographic Database. In *Proc. Intl. Conf. on Spatial Information Theory (COSIT)*, 1995.

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

[PSV96] C.H. Papadimitriou, D. Suciu, and V. Vianu. Topological queries in spatial databases. In *15th ACM Symp. on Principles of Database Systems*, Montréal, June 1996.

[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.

[PVV95] J. Paredaens, J. Van den Bussche, and D. Van Gucht. First-order queries on finite structures over the reals. In *Proceedings 10th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1995.

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

[Sei91] R. Seidel. A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and for Triangulating Polygons. *Computational Geometry*, 1(1), 1991.

[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; ftp://ftp.cnam.fr/pub/CNAM/cedric/tech_reports/RRC-96-09.ps.Z, Cedric Lab, CNAM, Paris, 1996.

[TvW88] R.E. Tarjan and C.J. van Wyk. An $O(n \log\log n)$ Time Algorithm for Triangulating a Simple Polygon. *SIAM Journal of Computing*, 17(1):143–178, 1988.

[VG96] V.Gaede and O. Guenther. Multidimensional Access Methods. Technical Report TR-96-043, ICSI, 1996.

[VGV95] L. Vandeurzen, M. Gyssens, and D. Van Gucht. On the desirability and limitations of linear spatial database models. In *Advances in Spatial Databases, 4th Int. Symp., SSD'95*, pages 14–28. Springer, 1995.