Indexing Constraint Databases by Using a Dual Representation

E. Bertino B. Catania
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
Via Comelico 39/41, 20135 Milano, Italy
e-mail: {bertino,catania}@dsi.unimi.it

B. Chidlovskii
Xerox Research Center Europe
6, chemin de Maupertuis
38240 Meylan France
e-mail: chidlovskii@xrce.xerox.com

Abstract

Linear constraint databases are a powerful framework to model spatial and temporal data. The use of constraint databases should be supported by access data structures that make effective use of secondary storage and reduce query processing time. Such structures should be able to store both finite and infinite objects and perform both containment (ALL) and intersection (EXIST) queries. As standard indexing techniques have certain limitations in satisfying such requirements, we employ the concept of geometric duality for designing new indexing techniques. In [5], we have used the dual transformation for polyhedra to develop a dynamic optimal indexing solution based on B$^+$-trees, to detect all objects contained in or intersecting a given half-plane, when the angular coefficient belongs to a predefined set. In this paper, we extend the previous solution to allow angular coefficients to take any value. We present two approximation techniques for the dual representation of spatial objects, based on B$^+$-trees. The techniques handle both finite and infinite objects and process both ALL and EXIST selections in a uniform way. We show the practical applicability of the proposed techniques by an experimental comparison with respect to R$^+$-trees.

1. Introduction

Constraint programming has been recognized as an attractive paradigm for database applications, because of its completely declarative nature and the ability to model spatial and temporal concepts at different database levels [4, 15]. At the data level, constraints allow the finite representation of possibly infinite sets of relational tuples. For example, the constraint $x \leq 2 \wedge y \geq 3$, where x and y are real variables, represents the infinite set of tuples having the x attribute equal or lower than 2 and the y attribute equal or greater than 3; such constraint is called *generalized tuple*. The set of solutions of a generalized tuple is called *extension* of the tuple. At the query language level, the integration of constraints into relational languages increases their expressive power [15], preserving at the same time all the good features of relational languages.

In order to efficiently deal with constraint databases, new indexing structures have to be defined with time and space complexities comparable to those of B$^+$-trees [9]. For 1-dimensional data, B$^+$-trees process range queries in $O(\log_B n + t)$ I/O operations, require $O(n)$ blocks of secondary storage, and perform insertions/deletions in $O(\log_B n)$ I/O operations. [1]

EXIST/ALL selections. When constraint databases are used to store spatial objects, at least two types of queries should be supported by indexing structures to support constraint language features: ALL selection, retrieving all generalized tuples whose extension is contained in the extension of a given generalized tuple, called *query generalized tuple*, and EXIST selection, retrieving all generalized tuples whose extension has a non-empty intersection with the extension of the query generalized tuple.

Some indexing techniques designed for constraint databases have been derived from R-tree, R$^+$-tree, quadtree or their variants [10, 14, 19]. They essentially support EXIST selections and assume that only finite objects are stored [3, 16, 18]. Some solutions are based on the approximation of the extension of generalized tuples [7, 11, 20] by bounded rectangular boxes; such approximation requires a refinement step to filter the tuples satisfying the query. Such techniques are not always suitable to support ALL selection. As an example, consider the R$^+$-tree which is a direct extension of B$^+$-tree to d dimensions [19] If the query object is not a rectangle, an ALL selection has to be approximated by an EXIST selection to guarantee the correctness of the execution. Clearly, the substitution of an ALL se-

[1] In the given complexity bounds, N is the number of items in the database; B is the number of items per disk block; T is the number of items in the problem solution; $n = N/B$ is the optimal number of blocks required to store the database; $t = T/B$ is the optimal number of blocks to access for reporting the problem result.

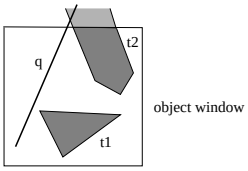

Figure 1. Approximation of unbounded polyhedra

lection with one EXIST selection is a rather poor solution, as the number of data objects a query object intersects is often considerably lower than the number of objects the query object contains.

Finite/infinite objects. Another attractive feature of an indexing mechanism for constraint databases is the storage of infinite (unbounded) objects, useful, for examples, for Operations Research applications [6]. Unfortunately, standard indexing structures support the representation of finite objects only. Infinite objects can be transformed into finite ones by imposing, when necessary, some global constraints. For example, all unbounded objects can be cut on the boundary of a certain "object window". However, the joint use of constraints inherent to the object itself with the global ones decreases the structure flexibility, since any change in global constraints would require to revisit all tuples in the database. Moreover, this approach is not always correct. Figure 1 shows an example of such approximation. The unbounded polyhedron t_2 and query q do not intersect inside the object window, even if they do outside it.

Duality. In [5], we have employed the concept of geometric duality to define an indexing structure which allows one to store and process both finite and infinite objects. The proposed technique is based on the dual representation of generalized tuples expressed in the linear polynomial constraint theory. In the proposed technique, a query is assumed to be a half-plane (also called *half-plane query*). In linear constraint databases, this kind of queries is very important since each inequality constraint, expressed by using the linear polynomial constraint theory, represents a half-plane. Such query is widely used by different applications in linear programming, CAD, VLSI, and spatial databases. We have shown that, under the previous conditions, by using the dual transformation for polyhedra presented in [13], ALL and EXIST selections can be reduced to a point location problem and, when the angular coefficient of the half-plane query belongs to a predefined set, a dynamic optimal indexing solution based on B$^+$-trees can be used for both selections. However, if the query angular coefficient does not belong to the predefined set, the logarithmic time complexity can be retained but the space complexity is no longer linear. The use of the dual representation for indexing constraint databases has also been recently considered in [1], to determine all points located above a given line.

Our contribution. In this paper, we propose two approximation techniques based on the dual representation of objects, which are essentially different from any approximation technique usually adopted for object representation. Both approximation techniques extend the B$^+$-tree solution presented in [5] assuming that the angular coefficient of the line associated with the query half-plane does not belong to a predefined set. We first show that d searches against d different B$^+$-trees are sufficient in E^d to approximate an arbitrary half-plane query. Since the results of the d searches are not necessarily disjoint, such approximation suffers from the duplication problem. To overcome this problem, we improve the technique and propose an approach by which only one search in one of the selected B$^+$-trees, say B_1, is performed and then the search to be performed on the other B$^+$-trees is approximated by an additional search in B_1. The sets of leaf nodes of B_1 scanned during the two searches are disjoint and therefore no duplicates are generated. In order to show the practical applicability of the proposed approximation technique, we also experimentally compare it with the R$^+$-tree organization, with respect to the 2-dimensional space.

The paper is organized as follows. Section 2 introduces constraint databases and the dual representation. Section 3 proposes an indexing technique based on the dual representation to solve a restricted EXIST/ALL problem. Approximation techniques are introduced in Section 4, whereas experimental results are discussed in Section 5. Finally, Section 6 presents some conclusions and outlines future work.

2 Constraint databases

A *linear constraint* in variables $x_1, ..., x_d$ is a formula of the form $a_1 x_1 + ... + a_d x_d + c\ \theta\ 0$, where coefficients a_i, $i = 1, ..., d$, are real numbers and $\theta \in \{=, \neq, \leq, <, \geq, >\}$. A conjunction of m linear constraints in variables $x_1, \ldots, x_d$ has the form $\wedge_{i=1}^{m} a_1^i x_1 + ... + a_d^i x_d + c^i\ \theta^i\ 0$, where $\theta^i \in \{=, \neq, \leq, <, \geq, >\}$. In this paper, we assume $\theta^i \in \{=, \leq, \geq\}$[2] and replace each equality constraint $a_1^i x_1 + ... + a_d^i x_d + c^i = 0$ by the equivalent conjunction of constraints $a_1^i x_1 + ... + a_d^i x_d + c^i \geq 0 \wedge a_1^i x_1 + ... + a_d^i x_d + c^i \leq 0$.

A conjunction of linear constraints of the form $\wedge_{i=1}^{m} a_1^i x_1 + ... + a_d^i x_d + c^i\ \theta^i\ 0$, where $\theta^i \in \{\leq, \geq\}$ is called *generalized tuple*. A set of generalized tuples forms a *generalized relation*. If each generalized tuple is interpreted as a set of points in a d-dimensional space, the constraint database can be seen as a spatial database.

In this paper, we are interested in two types of queries for constraint databases, EXIST and ALL selections. For a

[2] The approach can be easily extended to the case $\theta^i \in \{=, \neq, \leq, <, \geq, >\}$.

given generalized relation r, they are defined as follows:[3]

ALL selection. It retrieves all tuples in r whose extension is *contained* in the extension of a given tuple q, called *query tuple*. If the extension of tuple t is *contained* in the extension of the query tuple q, we denote this fact by ALL(q,t) and, given a relation r, we let ALL$(q,r) = \{t \in r |$ ALL$(t,q)\}$.

EXIST selection. It retrieves all tuples in r whose extension has a *non-empty intersection* with the extension of a query tuple q. If the extensions of t and q have a non-empty intersection, we denote this fact with EXIST(q,t) and, given a relation r, we let EXIST$(q,r) = \{t \in r |$EXIST$(t,q)\}$.

In the following, we analyze ALL and EXIST selections with respect to a *query half-plane* which has the form: $q \equiv a_1x_1 + ... + a_dx_d + c\ \theta\ 0$, where $\theta \in \{\geq, \leq\}$. Given a half-plane q, a relation r and $Q \in \{ALL, EXIST\}$, $Q(q,r)$, or $Q(q)$ when r is not specified, is called a *query* whereas Q is called the *type* of $Q(q,r)$.

By using a spatial terminology, we call *hyperplane* in a d-dimensional space (E^d) the spatial object defined by equation $a_1x_1 + ... + a_dx_d + c = 0$ and *half-plane* in a d-dimensional space the spatial object defined by equation $a_1x_1 + ... + a_dx_d + c\ \theta\ 0$, $\theta \in \{\geq, \leq\}$. A *d-dimensional convex polyhedron* P in a d-dimensional space is defined as the intersection of a finite number of half-planes. Moreover, we denote by $p(P)$ the boundary of P and with t_P the tuple representing P. The faces of P that are a subset of some supporting hyperplane with $\theta =$'$\geq$' and $a_d \leq 0$ ($\theta =$'$\geq$' and $a_d \geq 0$) form the *upper hull* (*lower hull*) of P;

2.1 The dual transformation

The dual transformation is a basic operation applied to d-dimensional objects in different geometrical algorithms [1, 8, 10, 13]. Here we assume that none of the considered half-planes is vertical.[4] Under this hypothesis, a hyperplane $a_1x_1 + ... + a_dx_d + c = 0$ intersects the d-th coordinate in a unique point represented by the equation:

$x_d = b_1x_1 + ... + b_{d-1}x_{d-1} + b_d$

where $b_i = -a_i/a_d, i = 1, ..., d-1$ and $b_d = c/a_d$. Given a hyperplane H, the function $F_H : E^{d-1} \rightarrow E^1$ is introduced as follows :

$F_H(x_1, ..., x_{d-1}) = b_1x_1 + ... + b_{d-1}x_{d-1} + b_d$.

A point $p = (p_1, ..., p_d)$ lies *above* (*on*, *below*) hyperplane H if $p_d >$ (=, <) $F_H(p_1, ..., p_{d-1})$. Using the dual transformation, each hyperplane is mapped into a point and vice versa. In particular, the dual representation $D(H)$ of a hyperplane $x_d = b_1x_1 + ... + b_{d-1}x_{d-1} + b_d$ is the point $(b_1, ..., b_d) \in E^d$. On the contrary, the dual representation $D(p)$ of a point $p = (p_1, ..., p_d)$ is the hyperplane defined by the equation $x_d = -p_1x_1 ... - p_{d-1}x_{d-1} + p_d$. We call *primal space* $\mathcal{P}$ the reference space of the original hyperplanes and *dual space* $\mathcal{D}$ the reference space of the dual representations. The dual transformation satisfies the following key property: *a point p lies above (on, below) a hyperplane H iff the dual $D(H)$ lies below (on, above) $D(p)$* [8, 10].

Polyhedra. The dual representation is extended to convex polyhedra by associating a pair of functions with each polyhedron [13]. If V_P denotes the set of vertices of a polyhedron P, the functions are defined as follows:

$TOP^P(x_1, .., x_{d-1}) = max_{v \in V_P}\{F_{D(v)}(x_1, ..., x_{d-1})\}$

$BOT^P(x_1, .., x_{d-1}) = min_{v \in V_P}\{F_{D(v)}(x_1, ..., x_{d-1})\}$.

These functions are piecewise linear and continuous. TOP^P is convex, whereas BOT^P is concave. Moreover, TOP^P (BOT^P) maps any slope $(b_1, ..., b_{d-1})$ of a non-vertical hyperplane into the maximum value b_d^{max} (b_d^{min}), such that the hyperplane $x_d = b_1x_1 + ... + b_{d-1}x_{d-1} + b_d^{max}$ ($x_d = b_1x_1 + ... + b_{d-1}x_{d-1} + b_d^{min}$) intersects P. It can be shown that each polyhedron O is associated with exactly one pair of functions TOP^P and BOT^P and vice versa. The two functions satisfy the following property.

Proposition 2.1 *For any point* $(p_1, ..., p_{d-1})$, $TOP^P(p_1, ..., p_{d-1}) \geq BOT^P(p_1, ..., p_{d-1})$. □

Unbounded polyhedra. If a polyhedron is not bounded (a common case in constraint databases), the definition of functions TOP^P and BOT^P should to be extended in order to deal with virtual vertices at infinity. Let C_P denote a d-dimensional cube that contains all vertices of P. Then, the bounded polyhedron $P \cap C_P$ has a set of vertices $V_{P \cap C_P} = V_P \cup \overline{V}$, where $\overline{V}$ contains those vertices that are formed by the intersections of C_P with edges of P. As the edge length of C_P goes to infinity, so do the vertices in $\overline{V}$. Thus, functions $TOP^P, BOT^P : E^{d-1} \rightarrow E^1 \cup \{+\infty, -\infty\}$ for unbounded polyhedra can be defined as follows:

$TOP^P(x_1, .., x_{d-1}) =$

$\quad lim_{e \rightarrow \infty} max_{v \in V_P \cup \overline{V}}\{F_{D(v)}(x_1, ..., x_{d-1})\}$

$BOT^P(x_1, .., x_{d-1}) =$

$\quad lim_{e \rightarrow \infty} min_{v \in V_P \cup \overline{V}}\{F_{D(v)}(x_1, ..., x_{d-1})\}$

There exists an isomorphism between the upper hull of a polyhedron P and the graph of TOP^P. Each k-dimensional face f of the upper hull of P corresponds to exactly one $(d-k-1)$-dimensional face $D(f)$ of TOP^P graph and vice versa. If two faces f_1 and f_2 of the P upper hull are adjacent, then so are $D(f_1)$ and $D(f_2)$. The same isomorphism exists between the lower hull of P and the graph of BOT^P. Consequently, if the number of vertices of P is n_v, the graphs of TOP^P and BOT^P are polyhedral

[3] In the remainder of the paper we consider *generalized tuples* and *generalized relations* only; for brevity, we will omit the adjective *generalized*.

[4] The proposed transformation can be extended to deal with vertical hyperplanes. We refer the reader to [13] for additional details.

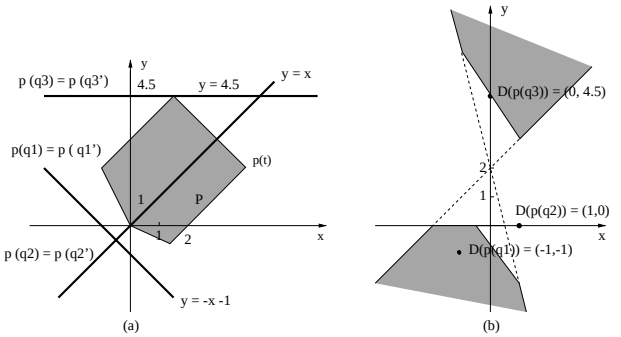

Figure 2. A finite polyhedron and some query half-planes: (a) in the primal plane $\mathcal{P}$; (b) in the dual plane $\mathcal{D}$

surfaces in E^d containing at most n_v $(d-1)$-dimensional faces and $O(n_v^2)$ $(d-2)$-dimensional faces.

By using the previous results and by considering tuples instead of polyhedra, we obtain the following result.

Proposition 2.2 *Let t_P be a tuple in relation r. Let $q(\theta)$ be the query half-plane $x_d\ \theta\ b_1x_1 + ... + b_{d-1}x_{d-1} + b_d$, where $\theta \in \{\geq, \leq\}$. Then:*

- *ALL$(q(\geq), t_P)$ iff $b_d \leq BOT^P(b_1, ..., b_{d-1})$;*
- *ALL$(q(\leq), t_P)$ iff $b_d \geq TOP^P(b_1, ..., b_{d-1})$;*
- *EXIST$(q(\geq), t_P)$ iff $b_d \leq TOP^P(b_1, ..., b_{d-1})$;*
- *EXIST$(q(\leq), t_P)$ iff $b_d \geq BOT^P(b_1, ..., b_{d-1})$.* □

Example 2.1 *Figure 2 presents an example of dual transformation. Given the half-plane queries $q_1 \equiv y \geq -x - 1$, $q_2 \equiv y \geq 4.5$, $q_3 \equiv y \geq x$, one can see in Figure 2(b) that $-1 < BOT^P(-1)$, $4.5 = TOP^P(0)$ and $BOT^P(1) < 0 < TOP^P(1)$. It follows from Proposition 2.2 that ALL(q_1, t), EXIST(q_2, t) and EXIST(q_3, t) are satisfied. Figure 2(a) confirms these results. Similarly, if we consider the queries $q_1' \equiv y \leq -x - 1$, $q_2' \equiv y \leq 4.5$ and $q_3' \equiv y \leq x$, it follows from Proposition 2.2 that ALL(q_2', t) and EXIST(q_3', t) are satisfied (see Figure 2(a)).* ◇

It has been shown that values $TOP^P(b_1, ..., b_{d-1})$ and $BOT^P(b_1, ..., b_{d-1})$ can be computed by a $d-1$-dimensional point location on the plane $b_d = 0$, with respect to the partition induced by the projection of the TOP^P (BOT^P) graph on plane $b_d = 0$. See [13] for details.

3 Restricted ALL/EXIST problem

In the previous section we have seen that the dual transformation uniformly reduces both ALL and EXIST selections to a point location problem. For this problem, as well as its multiple variants, some efficient in-memory algorithms have been proposed (see [17] for a survey). However, similar algorithms designed for the use in external storage are still far from being adequately efficient. Some efficient algorithms have been designed for 2- and 3-dimensional objects [2, 3, 12], but solutions for higher dimensions are not so efficient, especially with respect to space complexity [13].

In this section, we consider a special case of ALL and EXIST selections, which is typical of VLSI and CAD design. We assume that, given a query half-plane $x_d\ \theta\ b_1x_1 + ... + b_{d-1}x_{d-1} + b_d$, point $(b_1, ..., b_{d-1})$ belongs to a predefined set S. This assumption allows us to precompute TOP^P and BOT^P surface values for specific points and use them during a query execution.

Indeed, due to Proposition 2.2, in order to check intersection and containment between a set of polyhedra and half-plane $x_d\ \theta\ b_1x_1 + ... + b_{d-1}x_{d-1} + b_d$, it is sufficient to maintain two sets of values. Given a relation r, for each tuple $t_P \in r$, the first set contains value $TOP^P(b_1, ..., b_{d-1})$ whereas the second set contains value $BOT^P(b_1, ..., b_{d-1})$. Since both sets of points are totally ordered, they can be organized in two lists, denoted by B^{up} (for TOP^P values) and B^{down} (for BOT^P values), in the increasing order of values.[5] Given a query half-plane $q(\theta) \equiv x_d\ \theta\ b_1x_1 + ... + b_{d-1}x_{d-1} + b_d$, it is easy to see that the position of b_d in the total order determines the result of the query. Indeed:

- ALL$(q(\geq), r)$ is represented by all the tuples associated with points following or equal to b_d in B^{down}.
- ALL$(q(\leq), r)$ is represented by all the tuples associated with points preceding or equal to b_d in B^{up}.
- EXIST$(q(\geq), r)$ is represented by all the tuples associated with points following or equal to b_d in B^{up}.
- EXIST$(q(\leq), r)$ is represented by all the tuples associated with points preceding or equal to b_d in B^{down}.

In other words, to perform selections against a set of tuples in secondary storage, it is sufficient to maintain, for each point in S, two ordered sets of values; therefore, B$^+$-trees can be used to this purpose. Moreover, this solution provides a uniform approach to ALL and EXIST selections. Given a query $Q(x_d\ \theta\ b_1x_1 + ... + b_{d-1}x_{d-1} + b_d, r)$, $Q \in \{$ALL,EXIST$\}$, the search algorithm first selects the corresponding B$^+$-tree associated with point $(b_1, ..., b_{d-1})$, that is, B^{up} for queries ALL$(q(\leq), r)$ and EXIST$(q(\geq), r)$ and B^{down} for queries ALL$(q(\geq), r)$ and EXIST$(q(\leq), r)$. Then, value b_d is searched in the selected B$^+$-tree and all

[5] Infinite values are assumed to be finitely approximated and stored in the lists.

leaf values are swept in the direction corresponding to the query.[6]

Theorem 3.1 *Let r be a relation containing N tuples. Let $q \equiv x_d\ \theta\ b_1x_1 + ... + b_{d-1}x_{d-1} + b_d$ be a query half-plane. Let T be the cardinality of the set $ALL(q,r)$ (respectively $EXIST(q,r)$). If $(b_1,...,b_{d-1})$ is contained in a predefined set S of cardinality k, there is an indexing structure for storing r in $O(k\ N/B)$ pages such that $ALL(q,r)$ and $EXIST(q,r)$ selections are performed in $O(\log_B N/B + T/B)$ time, and tuple update operations are performed in $O(k\ \log_B N/B)$ time.* □

If $(b_1,...,b_{d-1}) \notin S$, a $d-1$-dimensional point location must be performed in order to compute $TOP^P(b_1,...,b_{d-1})$ and $BOT^P(b_1,...,b_{d-1})$ (see Section 2.1). If the relation r contains N tuples and each tuple represents a polyhedra with at most m vertices, the projection of the TOP^P (BOT^P) graphs, one for each tuple t_P, onto the plane $J \equiv b_d = 0$, generates N^2m^2 partitions. Any given partition M corresponds to a specific order of polyhedra. Moreover, any given partition M corresponds to a specific vertex of the upper hull of each polyhedron.

From the previous discussion, it follows that one obvious way to define a data structure to answer a general half-plane query requires maintaining one B$^+$-tree for each given partition of J, induced by TOP^P surface values, and one B$^+$-tree for each given partition of J, induced by BOT^P surface values. Then, given a half-plane $x_d\ \theta\ b_1x_1 + ... + b_{d-1}x_{d-1} + b_d$, a point location is performed to determine the partition containing point $(b_1,...,b_{d-1})$; the corresponding B$^+$-tree is then used to answer the query.

Obviously, such solution is unsatisfactory, as it leads to the $O(n^3m^2)$ space complexity. A possible solution is to apply an approach based on the approximation and post-filtering step, when $(b_1,...,b_{d-1}) \notin S$. The aim of the following section is to introduce such an approach.

4 Approximation of ALL/EXIST queries

The technique proposed in the previous section works for queries whose angular coefficient belongs to a predefined set S. To allow queries to take any angular coefficient, we design an approximation technique based on the dual representation. Unlike typical spatial approaches, where usually polyhedra are approximated with rectangular boxes, we store polyhedra without changes; however, half-plane queries are approximated with half-planes whose angular coefficients belong to the set S. To illustrate how the technique works, we first present two solutions in 2-dimensional space; then we show how to generalize the approach to a d-dimensional space, $d > 2$.

[6] Another solution to the same problem can be provided by reducing ALL and EXIST selections to the 1-dimensional interval management problem (see [5] for additional details).

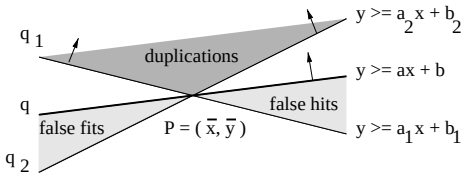

Figure 3. An approximation example

4.1 Half-plane query approximation

We denote with S a predefined set of angular coefficients, $a_i \in S, i = 1,..,k$ in E^2, and assume that all B^{up} and B^{down} B$^+$-trees described in Section 3 are constructed for all $a_i \in S, i = 1,..,k$. We consider a half-plane query $q \equiv y\ \theta\ ax + b$ in E^2, where $a \notin S$. The half-plane query can be approximated with two new half-plane queries (called *app-queries*), $q_1 \equiv y\ \theta_1\ a_1x + b_1$ and $q_2 \equiv y\ \theta_2\ a_2x + b_2$, where $a_1, a_2 \in S$ and the union of q_1 and q_2 covers the original query half-space. The latter feature guarantees that any tuple in the result of query q appears in the result of at least one app-query, ensuring the *correctness* of approximation (see Figure 3).

The approximation of a half-plane with the union of two new half-planes raises two main issues (see Figure 3):

Duplications: as two app-queries may intersect, some tuples are returned twice.

False fits: not all the tuples satisfying the app-queries satisfy also the original one, thus a refinement step is required to discard such *false hits*.

The number of duplicates and false hits produced by the approximation depends on the choice of app-queries. Moreover, the problems of duplications and false hits appear to be complementary: app-query pairs reducing the number of false hits often increase the number of duplicates and vice versa. In the following, we minimize false fits rather than duplications. Indeed, we argue that the generation of duplicates is generally less painful than the retrieval of tuples which do not satisfy the query at all. In Section 4.2 we will improve the approximation to keep low both duplications and false fits.

For a given query q, the optimal choice of the app-queries q_1 and q_2, is performed by choosing: (1) the lines associated with q_1 and q_2 (coefficients a_1, a_2, b_1, b_2), (2) operators θ_1, θ_2 and (3) query type (EXIST/ALL), which guarantee the correct approximation and minimize the number of false hits. All these choices will be discussed by considering a query with respect to a half-plane $y \geq ax + b$

Conditions on a, a_1, a_2	Values for θ_1 and θ_2
$a_1 < a < a_2$	$\theta_1 \equiv \theta, \theta_2 \equiv \theta$
$a_1 < a, a_2 < a$	$\theta_1 \equiv \theta, \theta_2 \equiv \neg\theta$
$a < a_1, a < a_2$	$\theta_1 \equiv \neg\theta, \theta_2 \equiv \theta$

Table 1. Choice of the half-plane app-queries

(similar conditions can be given for queries with respect to $y \leq ax + b$).

Choice of the angular coefficients a_1, a_2. In order to minimize false hits, the lines associated with q_1 and q_2 must be chosen in such a way that the false hit area would contain none or few tuples. If the tuple distribution in E^2 is close to uniform or unknown, this task is equivalent to minimize the size of the false hit area. Thus, we choose coefficients a_1 and a_2 from S which are nearest to a; we denote with a_1 (a_2) the angular coefficient in S which is encountered by performing a clockwise (anti-clockwise, respectively) rotation of line $y = ax + b$ (see Figure 3).

Choice of b_1, b_2. Given the angular coefficients a_1 and a_2, coefficients b_1 and b_2 are determined by choosing a point P on line $y = ax + b$ and making $p(q_1)$ and $p(q_2)$ passing for P. The optimal choice of P depends on the tuple distribution on the plane. We omit details due to space limitations.

Choice of the half-planes (θ_1 and θ_2). Given the lines for app-queries q_1 and q_2, we select the operators θ_1 and θ_2 in such a way that the union of the points belonging to q_1 and q_2 covers the original query half-plane. Three possible cases may arise. Table 1 shows the values for θ_1 and θ_2 for each possible combination of a, a_1 and a_2. In the table, $\neg\theta$ corresponds to '$\leq$' if θ is '$\geq$' and '$\geq$' if θ is '$\leq$'.

Type of the app-queries. Finally, a proper type should be assigned to app-queries q_1 and q_2 to guarantee the correct approximation; the types are derived from the type of the original query q:

- EXIST *query*: the approximation of an original EXIST query with two EXIST app-queries is correct; indeed, each tuple satisfying the original query is returned by at least one of the app-queries.

- ALL *query*: the replacement of an original ALL query with two ALL app-queries may not be correct. Figure 4 shows an example where one tuple is contained in the original query half-plane but is not contained in any of the ALL app-queries. To preserve correctness, like what happens with R^+-tree, we approximate an ALL query with one EXIST and one ALL app-query. It is simple to show that this approximation is correct.

Since the angular coefficients of the new app-queries belong to S, the technique presented in Section 3 can be applied to execute them. Thus, we obtain the following result.

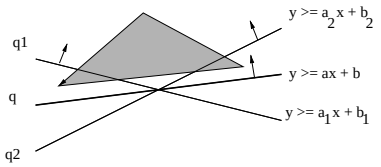

Figure 4. Choice of the type of app-queries for an original ALL query

Theorem 4.1 *Let r be a relation containing N tuples. Let q be a half-plane query. Let T be the cardinality of the set $ALL(r, q)$ ($EXIST(r, q)$). If the angular coefficient of $p(q)$ is not contained in a predefined set of cardinality k, there is an indexing structure for storing r in $O(k\ N/B)$ pages such that $ALL(r, q)$ and $EXIST(r, q)$ selections are performed in $O(log_B N/B + T_1/B + T_2/B)$ time, where T_1 and T_2 represent the number of tuples returned by the two new app-queries q_1 and q_2 generated as above, and tuple update operations are performed in $O(k log_B N/B)$ time.* □

4.2 Approximation by extending B$^+$-trees

By using the approximation technique presented in the previous subsection (hereafter denoted by T1) some result tuples may be returned twice. One possible solution to avoid the duplication problem is to search in one of the B$^+$-trees only, say B_1, and then approximate the search to be performed on the other B$^+$-tree, B_2, by an additional search in B_1. If the two searches return disjoint sets and starts from the same value, they can be combined into one and no duplicates are generated. However, to correctly approximate the search in B_2 with a search in B_1, some additional information about the tuples has to be inserted into the leaf nodes of B_1.

In the following, we develop the second approximation technique (denoted by T2) for both EXIST and ALL half-plane queries, assuming a set S of angular coefficients is predefined. From Table 1, we can see that, by applying technique T1, three cases may arise in the approximation of query $Q(y\theta ax + b)$ with two app-queries $Q_1(y\theta_1 a_1 x + b_1)$ and $Q_2(y\theta_2 a_2 x + b_2)$: 1) $a_1 < a < a_2$; 2) $a > a_1, a > a_2$ and 3) $a < a_1, a < a_2$. Due to the space limitation, in the following we discuss technique T2 only for the main case $a_1 < a < a_2$. The other cases are similarly handled.

4.2.1 EXIST queries

In the following, we describe technique T2 for queries EXIST($y \geq ax + b$). The symmetric case EXIST($y \leq ax + b$) is similarly handled.

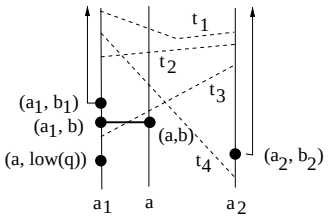

Figure 5. Searches in B_1^{up} and B_2^{up} for the query EXIST$(y \geq ax + b)$

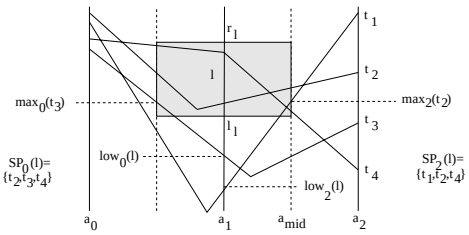

Figure 6. Handicap values associated with leaf l in B_1^{up}

Under the hypothesis $a_1 < a < a_2$, technique T1 approximates a query EXIST$(y \geq ax + b)$ with two queries EXIST$(y \geq a_i x + b_i), i = 1, 2$. The execution of query EXIST$(y \geq a_i x + b_i)$, $i = 1, 2$, is supported by a B$^+$-tree (see Section 3), hereafter denoted by B_i^{up}, storing the intersection points of TOP^P with line $x = a_i$, $i = 1, 2$, for each tuple t_P contained in the relation. To execute query EXIST$(y \geq a_i x + b_i)$ on B_i^{up}, value b_i is searched in B_i^{up} and all values greater than b_i are retrieved by sweeping all leaf nodes in B_i^{up} starting from the node containing b_i. Such an approach is graphically represented in Figure 5; arrows show the upward direction of the sweeping process.

The leaf sweeping routines in B_1^{up} and B_2^{up} often encounter the same tuples (tuples t_1 and t_2 in Figure 5), generating numerous duplications. In the new approach, we approximate the original query with a single B$^+$-tree search; this allows to avoid duplications because any TOP^P intersects line $x = a_i$, $i = 1, 2$ only once and the search accesses a leaf node only once, too. The B$^+$-tree chosen for the single search is the one associated with the angular coefficient $a_i \in S$ such that the order of values TOP^P in B_i^{up} is less divergent from the order of values TOP^P on line $x = a$. Thus, we choose the nearest angular coefficient to a. Without loss of generality, we assume that a_1 is closer to a than a_2, i.e., $|a_1 - a| < |a_2 - a|$, and consider the query approximation with B_1^{up}.

While the sweeping direction is clearly upward oriented, the starting point choice is not obvious. If we start from the nearest projection point (a_1, b), the sweep can miss some tuples (like tuple t_3 in Figure 5). Starting from b_1 or b_2 is not correct either. Actually, the sweeping routine should start from some low point $low(q)$ on line $x = a_1$, where $low(q) \leq b$. To guarantee correctness, a search starting from $low(q)$ has to retrieve at least all the tuples t_P whose values TOP^P intersect $x = a_1$ in a point lower than b and intersect the line segment which joins the query point (a, b) and its projection (a_1, b) on $x = a_1$ (see tuple t_3 in Figure 5). In the following, we show how to calculate and store the handicap values and use them in the search.

Step 1: Handicap values. We start by computing a handicap value for each leaf node in B_1^{up}. First, we consider the vertical strip in $\mathcal{D}$ limited by lines $x = a_1$ and $x = a_{mid}$, where $a_{mid} = (a_1 + a_2)/2$; here a_{mid} corresponds to the worst approximation of query EXIST$(y \geq ax + b)$ with one search in B_1^{up}. Second, we partition the tuples t_P in the strip between leaf nodes of B_1^{up} in the way that tuples associated with a leaf allow to calculate the leaf handicap value.

Tuples are associated with leaf nodes as follows. For each tuple t_P, we calculate the maximum value t_2^{max} achieved by TOP^P in the range $[a_1, a_{mid}]$.Then, for a given leaf node l, let $SP_2(l) = \{t | l_l \leq t_2^{max} \leq r_l\}$, where $[l_l, r_l]$ is the smallest interval containing all key values in l. Then, the lowest intersection of tuples in $SP_2(l)$ with $x = a_1$ is chosen as the handicap value $low_2(l)$: $low_2(l) = min\{TOP^P(a_1) | t \in SP_2(l)\}$, where $TOP^P(a_1)$ is the intersection point of TOP^P with $x = a_1$.

Step 2: Associating new values with leaf nodes. Once we have calculated the handicap value $low_2(l)$ for a leaf l in B_1^{up}, we store it in the node. Generally, a leaf of B_1^{up} is extended with two handicap values: $low_2(l)$ for queries with the angular coefficient a, $a_1 \geq a \geq (a_1 + a_2)/2$, and, symmetrically, value $low_0(l)$ for the case $(a_0 + a_1)/2 < a \geq a_1$, where a_0 is the value preceding a_1 in the increasing ordering of S.

Figure 6 shows the values $low_0(l)$ and $low_2(l)$ calculated for a leaf node l in B_1^{up}. Note that the handicap values can be computed during the preprocessing phase, in a time which is linear in the dimension of relation r. Also, it can be shown that any update in a B$^+$-tree requires $O(log_B n)$ amortized time, because the removal/insertion of a leaf value rarely influences handicap values. Indeed, $O(n)$ leaf nodes in a B$^+$-tree store $O(N)$ TOP^P surface values and one leaf node has $O(1)$ handicap values.Therefore, only $O(n)$ tuples contribute in handicap values and, on average, B update operations result in one handicap value change, where B is the number of items per node. To perform such a change, we can properly extend the internal nodes of the B$^+$-tree in order to trace the links between the node storing a tuple, and the nodes the tuple imposes handicap values on. Thus, any tuple update takes $O(log_B n)$ amortized time.

Step 3: Performing the search. Once the leaf nodes are

extended with handicap values, the search is performed as follows. Given a query hyperplane $q \equiv y \geq ax + b$, $a_1 < a < a_{mid}$, we first perform the upward leaf sweeping in B_1^{up} starting from (a_1, b), check values $low_2(l')$ for all the encountered leaf nodes l' and keep the lowest of them: $low(q) = min\{low_2(l') | l' \in$ first sweeping in $B_1^{up}\}$.

Once the upward sweeping routine is over, the value $low(q)$ indicates how far we should descend along $x = a_1$ to complete the retrieval of tuples satisfying the original query. Then a downward oriented sweeping starts from point (a_1, b) and ends in the leaf node containing value $low(q)$. Totally, the two leaf sweepings (upward and downward) in B_1^{up} retrieve a superset of EXIST(q, r). The correctness of the approximation follows from the fact that, starting the search from (a_1, b) and due to the computation of $low_2(l)$ values, the computation of $low(q)$ is performed by considering all tuples t_P such that $TOP^P(a) \geq b$, not considered by the first search. Although the two sweepings can retrieve false fits, the search is free of duplications.

The approximation technique we have described for queries EXIST$(y \geq ax + b)$ can be easily applied to queries EXIST$(y \leq ax + b)$. In this case, the B$^+$-trees B_i^{down} chosen for the approximation contain BOT^P values, starting sweeping steps are downward oriented, and values $high_0(l)$ and $high_2(l)$ are stored in each leaf l. From the results presented above, the following result holds.

Theorem 4.2 *Let r be a relation containing N tuples and q be a query half-plane. Let T be the cardinality of the set EXIST(q, r). If the angular coefficient of $p(q)$ is not contained in a predefined set S of cardinality k, there is an indexing structure for storing r in $O(k\ N/B)$ pages such that EXIST(q, r) selection is performed in $O(\log_B N/B + T_1/B)$ time, where T_1 is the number of tuples returned by the approximated query generated as above, and tuple update operations are performed in $O(k\ \log_B N/B)$ amortized time.* □

4.3 ALL queries

Consider the query ALL$(y \geq ax + b)$, with queries ALL$(y \leq ax + b)$ processed in a similar way. With the T1 technique (see Section 4.1), such ALL query can be approximated with one ALL and one EXIST query, respectively supported, for example, by B$^+$-trees B_1^{down} and B_2^{up}.In the new technique, if $a \notin S$, $a_1 < a < a_2$, and a_1 is closer to a than to a_2, we use B$^+$-tree B_1^{down} for the approximation. As the leaf node sweeping in B_1^{down} goes upward for the ALL query, we need handicap values $low_2(l)$ to be stored in leaf nodes of B_1^{down}, in addition to the handicap values $high_2(l)$ calculated for queries EXIST$(q(\leq))$. In this case, handicap values are calculated with respect to TOP^P surface values, due to the approximation of the second search with the query EXIST$(y \leq ax + b)$.

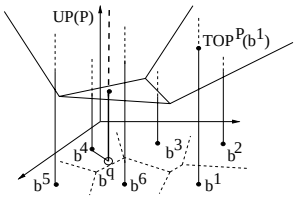

Figure 7. Example of approximation in E^3 (dual space)

Similar to the extension of leaf nodes in B_1^{down} with handicap values $low_0(l)$ and $low_2(l)$ for the approximation of queries ALL$(q(\geq))$, leaf nodes in B$^+$-tree B_1^{up} store handicap values $high_0(l)$ and $high_2(l)$ for the approximation of queries ALL$(q(\leq))$. In total, each leaf node in B$^+$-trees B_i^{up} and B_i^{down} is extended with four handicap values to support the approximation of all queries. For example, in a leaf node l of a B$^+$-tree B_i^{up}, the following values have to be stored : $low_{i+1}(l)$ and $low_{i-1}(l)$ for EXIST$(q(\geq))$ queries and $high_{i+1}(l)$ and $high_{i-1}(l)$ for ALL$(q(\leq))$ queries.

4.4 Extension to an arbitrary d-dimensional space

In E^d, each point $b^i = (b_1^i, ..., b_{d-1}^i) \in S$ represents a line in the dual representation of d-dimensional polyhedra. For any $b^i \in S$, two B$^+$-trees maintain values $TOP^P(b^i)$ and $BOT^P(b^i)$ for tuples t_P in relation r, and each B$^+$-tree stores at most N values (see Section 3). Given a query half-plane $x_d\ \theta\ b_1^q x_1 + ... + b_{d-1}^q x_{d-1} + b_d^q$, if point $b^q = (b_1^q, ..., b_{d-1}^q)$ belongs to the predefined set S, the corresponding B$^+$-tree has to be accessed to retrieve all tuples satisfying the query. If point $b^q \notin S$, the approximation is necessary. Below we revisit all aspects important for the choice of the approximation queries for q.

Choice of the slopes and hyperplanes. Unlike the 2-dimensional case where angular coefficients are ordered, in the d-dimensional space there is no order between points of S. If we are free to choose points $b^i \in S$, $i = 1, ..., k$, from E^{d-1} for the better approximation of queries, we choose them such that their distribution in E^{d-1} provides the smallest distance between an arbitrary b^q and the nearest b^i in S (see Figure 7).

To preserve the correctness of the approximation, technique T1 requires at most d searches in different B$^+$-trees. Instead, technique T2 requires just one approximation search, in the B$^+$-tree corresponding to the nearest point to the query point. To detect the point $b^i \in S$ nearest to b^q, we need the proximity partition of E^{d-1} induced, for example, by the Voronoi diagram [10] from the points of S (see Figure 7). If the cell corresponding to point $b^i \in S$ has d edges, each leaf node in B$^+$-trees B_i^{up} and B_i^{down} should

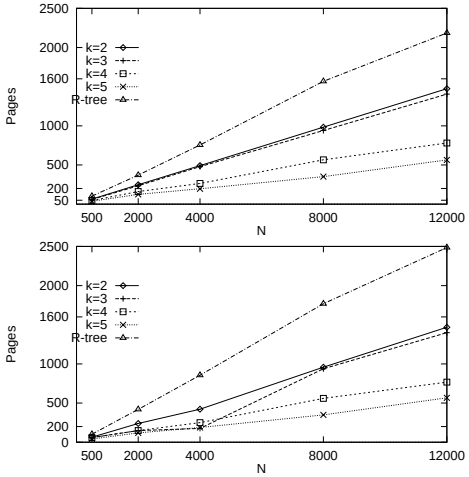

Figure 8. Performance of technique T2 and the R$^+$-tree on small objects. (a) EXIST selections; (b) ALL selections

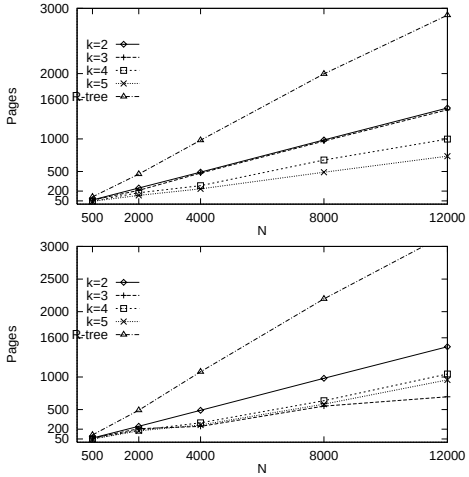

Figure 9. Performance of technique T2 and the R$^+$-tree on medium objects. (a) EXIST selections; (b) ALL selections

be extended with $4d$ handicap values. The calculation of the handicap values is similar to the 2-dimensional case. As the predefined set S does not change over time, the Voronoi diagram for proximity partition of E^{d-1} can be precomputed along with the handicap values.

Choice of the half-plane and query types. Rules proposed for the 2-dimensional case are still valid for the d-dimensional space, $d > 2$.

5 Experimental results

We have implemented technique T2 and we have compared its performance with respect to the R$^+$-tree performance [19] in the 2-dimensional space. Since R$^+$-tree can store bounded objects only, we have tested the two structure on the same sets of bounded polyhedra.

All experiments have been performed on a PC Pentium 133, under the Linux OS. Each stored value in a B$^+$/R$^+$-tree takes 4 bytes with the page size fixed to 1024 bytes. In the experiments, each (satisfiable) tuple is a conjunction of 3 to 6 linear constraints, with angular coefficients of constraints assigned randomly from the interval [0,$\pi/2$[$\cup$]$\pi/2$, π[. All tuples (their weight-center points) were uniformly distributed in the working window [-50:50, -50:50]. For any set of generated tuples, the bounding rectangle R containing all tuples has been detected. The parameters we have varied in the experiments are the following.

Object average size: Two different groups of relations have been considered, all containing finite objects. The first group contains *small* rectangles, i.e., rectangles whose area occupy 1-5% of the rectangle R area; the second group deals with *medium* rectangles, i.e., rectangles whose area does not exceed half the area of R. Since spatial databases typically deal with small objects, the size of the considered objects is a good parameter to analyze how the performance of R$^+$-trees changes by changing the average size of the considered objects.

Relation cardinality: Five different groups of relations have been considered, containing respectively 500, 2000, 4000, 8000, and 12000 tuples.

Query selectivity: We have considered six ALL queries and six EXIST (different from ALL) queries with the selectivities varying in the range 5-60%. In the following, we only report results obtained for the average range 10-15%. Performance results obtained for other selectivities appeared to be similar.

Cardinality of the angular coefficient set S: We have considered $k = 2, 3, 4, 5$.

The previous parameters have been combined in all possible ways. The results reported for the query selectivity 10-15% show that technique T2 performs better than R$^+$-trees in all cases, but at the cost of a higher space occupation. The performance results presented in Figures 8-10 are self-explanatory; here we summarize the main results.

Time complexity. As we can see from Figures 8 for small objects, and from Figures 9 for medium objects, technique T2 always performs better than the R$^+$-tree. From the same figures, we can also see that the R$^+$-tree performs better with small objects, whereas the behavior of technique T2 does not significantly change when the object size changes. Also, the advantage of technique T2 over the R$^+$-tree is wider for ALL selections.

Space complexity. The space complexity does not depend on the object average size. As we can see from Figure 10,

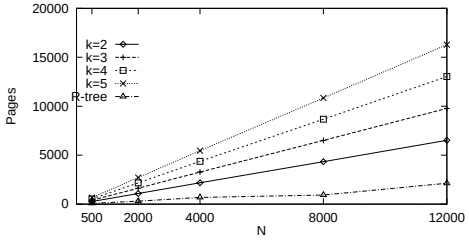

Figure 10. Disk space occupied by technique T2 and the R$^+$-tree

the space overhead for technique T2 is higher than that for the R$^+$-tree. On the average, for all values $k = 2, 3, 4, 5$, the B$^+$-trees used in technique T2 require the space occupied by R$^+$-tree multiplied by $1.32k$.

6 Concluding remarks

The paper has presented an approximation technique to determine all tuples in a constraint databases, or equivalently all finite and infinite polyhedra in a spatial database, that are intersected or contained in a given half-plane. The technique is based on a dual representation for polyhedra. Such technique extends the solution presented in [5] to deal with arbitrary half-plane queries. The proposed technique is based on the use of B$^+$-trees and can be applied to d-dimensional objects. The technique has been experimentally compared with R$^+$-trees, showing very promising performance.

All our experiments were conducted in E^2. Future work includes the analysis of the proposed technique in a d-dimensional space, $d > 2$. We argue that, in this case, the advantages in performance of our technique with respect to R$^+$-trees are even better. Indeed, by increasing the dimension of the space, the performance of our technique does not change, since we always deal with single values, whereas the R$^+$-trees performance decreases. Other implementations of the proposed techniques have also to be devised, in order to reduce the space occupancy, especially when considering a d-dimensional space, $d > 2$.

References

[1] P. Agarwal, L. Arge, J. Erickson, P. Franciosa, and J. Vitter. Efficient Searching with Linear Constraints. In *Proc. of PODS*, pages 169–178, 1998.

[2] L. Arge, D. Vengroff, and J. S. Vitter. External-Memory Algorithms for Processing Line Segments in Geographic Information Systems. In *Proc. of the 3rd Annual European Symp. on Algorithms*, pages 295–310, 1995.

[3] L. Arge, D. Vengroff, and J. S. Vitter. L. Arge and J. S. Vitter. Optimal Dynamic Interval Management in External Memory. In *Proc. of Foundations of Computer Science Conf.*, pages 560–569, 1996.

[4] A. Belussi, E. Bertino, and B. Catania. Manipulating Spatial Data in Constraint Databases. In *LNCS 1262: Proc. of the 5th Symp. on Spatial Databases*, pages 115–141, 1997.

[5] E. Bertino, B. Catania, and B. Shidlovsky. Towards Optimal Two-Dimensional Indexing for Constraint Databases. *Information Processing Letters*, 64(1):1–8, 1997.

[6] A. Brodsky, J. Jaffar, and M. Maher. Toward Practical Constraint Databases. In *Proc. of the 19th Int. Conf. on Very Large Data Bases*, pages 567–579, 1993.

[7] A. Brodsky, C. Lassez, J. Lassez, and M. Maher. Separability of Polyhedra and a New Approach to Spatial Storage. In *Proc. of PODS*, pages 7–11, 1995.

[8] K. Brown. *Geometric Transformations for Fast Geometric Algorithms*. PhD thesis, Carnegie-Mellon University, Pittsburgh, Pa., 1979.

[9] D. Comer. The Ubiquitous B-tree. *Computing Surveys*, 11(2):121–138, 1979.

[10] H. Edelsbrunner. Algorithms in Combinatorial Geometry. In *EATCS Monographs on Theoretical Computer Science, Vol. 10*. 1987.

[11] D. Goldin and P. Kanellakis. On Similarity Queries for Time-Series Data: Constraint Specification and Implementation. In *LNCS 976: Proc. of the First Int. Conf. on Principles and Practice of Constraint Programming*, pages 137–153, 1995.

[12] M. Goodrich, J.-J. Tsay, D. Vengroff, and J. Vitter. External-Memory Computational Geometry. In *Proc. of Found. of Computer Science Conf. (FOCS)*, pages 714–723, 1993.

[13] O. Günther. *Efficient Structures for Geometric Data Management*. Springer Verlag, 1988.

[14] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[15] P. Kanellakis, G. Kuper, and P. Revesz. Constraint Query Languages. *J. of Comp. and Syst. Sciences*, 51(1):26–52, 1995.

[16] P. Kanellakis, S. Ramaswamy, D. Vengroff, and J. Vitter. Indexing for Data Models with Constraints and Classes. In *Proc. of PODS*, pages 233–243, 1993.

[17] P.K. Agarwal and J. Erickson. Geometric Range Searching and its Relatives. In B. Chazelle, E. Goodman, and R. Pollack, editors, *Discrete and Computational Geometry: Ten Years Later*. American Mathematical Society, Providence, 1998.

[18] S. Ramaswamy. Efficient Indexing for Constraints and Temporal Databases. In *LNCS 1186: Proc. of the Int. Conf. Database Theory*, pages 419–431, 1997.

[19] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: A Dynamic Index for Multi-dimensional Objects. In *Proc. of the Int. Conf. on Very Large Data Bases*, pages 507–518, 1987.

[20] D. Srivastava. Subsumption and Indexing in Constraint Query Languages with Linear Arithmetic Constraints. *Annals of Mathematics and Artificial Intelligence*, 8(3-4):315–343, 1993.