Approximation Techniques for Indexing Two-Dimensional Constraint Databases

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

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

Abstract

Constraint databases have recently been proposed as a powerful framework to model and retrieve spatial 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. In this paper, we consider the indexing problem for objects represented by conjunctions of two-variable linear constraints and we analyze the problem of determining all generalized tuples whose extension intersects or is contained in the extension of a given half-plane. In [4], we have shown that both selection problems can be reduced to a point location problem by using a dual transformation [4, 10]. If the angular coefficient of the half-plane belongs to a predefined set, we have proved that a dynamic optimal indexing solution, based on B$^+$-trees, exists. In this paper we propose two approximation techniques that can be used to find the result when the angular coefficient does not belong to the predefined set. We also experimentally compare the proposed techniques with R-trees.

1 Introduction

Constraint programming is very attractive from a database point of view since it is completely declarative and since it allows to homogeneously model spatial and temporal concepts [3, 12]. Constraints can be added to relational database systems at different levels. At the data level, they finitely represent possibly infinite sets of relational tuples. For example, the constraint $2x + y \leq 2$, where x and y are real variables, represents all the points $(\overline{x}, \overline{y})$ such that $2\overline{x} + \overline{y} \leq 2$. A conjunction of constraints is called *generalized tuple*. The set of solutions of a generalized tuple is called *extension* of the generalized tuple. At the query language level, constraints increase the expressive power of simple relational languages by allowing mathematical computations.

Constraint databases should preserve all the good features of relational databases. In particular, new data structures should be defined for querying and updating constraint databases with time and space comparable to those of B$^+$-trees [7]. 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]

Since constraint databases are often used to store spatial objects, at least two constraint language features should be supported by index structures: ALL selection, retrieving all generalized tuples whose extension is contained in the extension of a given generalized tuple; EXIST selection, retrieving all generalized tuples whose extension has a nonempty intersection with the extension of a given generalized tuple.

Most of the existing indexing techniques for constraint databases, with optimal worst-case performance, only support EXIST selection and assume that index values are intervals [2, 13, 16]. Some other solutions are based on the approximation of the extension of generalized tuples (which must be closed) [5, 18] and on the use of some typical spatial data structures (such as R-trees and R$^+$-trees [11, 17]). The aim of this paper is to propose a simple indexing mechanism for both EXIST and ALL selections against two-dimensional linear constraints, representing both closed and open objects. The query generalized tuple is assumed to be a half-plane (also called *query half-plane*). In linear constraint databases this kind of query is very important since each inequality constraint, expressed by using the linear polynomial constraint theory, represents a half-plane. From an application point of view, this query is often used in linear programming problems and in spatial database applications.

[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.

In [4], we have shown how ALL and EXIST selections with respect to a query half-plane can be uniformly reduced to a point location problem [15] by using the concept of geometric duality [8]. If the angular coefficient of the line associated with a query half-plane belongs to a fixed set of cardinality k (*simplified problem*), we have proposed a solution in external storage, based on B$^+$-trees, requiring $O(k\ n)$ pages, performing ALL and EXIST selections in $O(\log_B n + t)$ and updates in $O(k\ \log_B n)$ I/O operations [4].

In this paper, we go one step forwards and, if the angular coefficient of the line associated with a query half-plane does not belong to the predefined set, we use the dual representation to define two approximation strategies, based on the technique we have defined to solve the simplified problem. The first technique we propose replaces the original query with two new queries. The second technique replaces the original query with a single new query. The first technique can be always applied, whatever the database. The second technique can be applied only if the database satisfies some specific requirements. The two techniques are then experimentally compared with R-trees, a well known spatial data structure [11].

The paper is organized as follows. Section 2 introduces constraint databases and the dual transformation. Section 3 proposes an indexing technique based on this representation to solve the simplified 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 and dual transformation

Constraint databases. A two-dimensional *linear constraint* in variables x and y is a formula of the form $a_1x + a_2y + c\ \theta\ 0$, where coefficients a_1, a_2, c are real numbers and $\theta \in \{=, \neq, \leq, <, \geq, >\}$. A conjunction of m two-dimensional linear constraints in variables x, y has the form $\wedge_{i=1}^{m} a_1^i x + a_2^i y + c^i\ \theta^i\ 0$, where $\theta^i \in \{=, \neq, \leq, <, \geq, >\}$ and is called *generalized tuple*. In this paper, we assume $\theta^i \in \{=, \leq, \geq\}$[2] and replace each equality constraint $a_1^i x + a_2^i y + c^i = 0$ by the equivalent conjunction of constraints $a_1^i x + a_2^i y + c^i \geq 0 \wedge a_1^i x + a_2^i y + c^i \leq 0$. A constraint $y \geq a_i x + b_i$ ($y \leq a_i x + b_i$) is called *down-constraint* (*up-constraint*). A generalized tuple is *satisfiable* if it admits at least one solution. A set of generalized tuples forms a *generalized relation*. If each generalized tuple is interpreted as a set of points in the 2-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 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 t is *contained* in the extension of q, we denote this fact by ALL(q, t) and, given a relation r, we let ALL$(q, r) = \{t \in r |\ \text{ALL}(q, t)\}$.

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$(q, r) = \{t \in r | \text{EXIST}(q, t)\}$.

In the following, we analyze ALL and EXIST selections with respect to a non-vertical *query half-plane* which has the form $y \theta ax + b$, where $\theta \equiv$'$\geq$' (*down-query*) or $\theta \equiv$'$\leq$' (*up-query*). 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 $Q(q, r)$. We denote with $p(t)$ ($p(C)$) the boundary of the extension of a tuple t (a constraint C). Given a constraint C and a tuple t, we say that C is *redundant* with respect to t if the tuple $t' = t \wedge C$ is equivalent to t. Satisfiable tuples of the type described above, without redundant constraints, and relations containing only such tuples, are called *regular*. In the following, since we deal only with regular tuples and relations, we omit the word "regular".

Dual transformation. Duality is a fundamental geometric method widely used in various domains [8, 10]. Let l be a line $y = ax + b$ in the *primal plane* $\mathcal{P}$. In the *dual plane* $\mathcal{D}$, l is transformed in point $D(l) = (a, b)$. Due to the same transformation, a point p in $\mathcal{P}$ is transformed in a line in $\mathcal{D}$. If the point p in $\mathcal{P}$ is (a, b), the corresponding dual line $D(p)$ is $y = -ax + b$.[4] Such transformation can be extended to deal with a tuple t by transforming each constraint of t into a point and connecting such points depending on the type of the constraint (up- or down-) and the type of the object representing the extension of the tuple (closed or open). Each tuple t is then transformed in a pair of disjoint and convex open domains, whose boundaries are an upward open ($UP(t)$) and a downward open ($DOWN(t)$) polygons (see Figure 1(b)), such that $DOWN(t)$ is *under* $UP(t)$.[5] Due to space constraints, we refer the reader to [4, 10] for additional details about the construction of the dual representation.

Since the boundary of a query half-plane corresponds to a point in $\mathcal{D}$, the dual transformation reduces both ALL and EXIST selections to a *point location problem* in plane $\mathcal{D}$.

Proposition 1 *[4] Let t be a tuple. Let $q(\theta)$ be a query tuple*

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

[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 dual transformation cannot be applied to vertical lines, therefore we assume that tuples do not contain constraints like $x \leq a$ or $x \geq a$.

[5] This means that, if a vertical line $x = a$ in $\mathcal{D}$ intersects both $DOWN(t)$ and $UP(t)$, the point of intersection with the former is always under the point of the intersection with the latter.

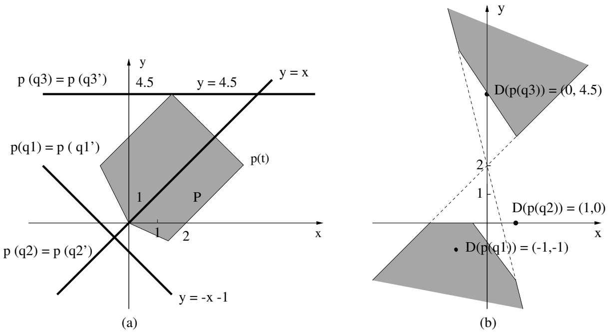

Figure 1. A tuple and some query examples in: (a) the primal plane $\mathcal{P}$, (b) the dual plane $\mathcal{D}$

$y \ \theta \ ax + b$, where $\theta \in \{\geq, \leq\}$. Then:
- ALL($q(\geq), t$) iff $b \leq DOWN(t)(a)$;[6]
- ALL($q(\leq), t$) iff $b \geq UP(t)(a)$;
- EXIST($q(\geq), t$) iff $b \leq UP(t)(a)$;
- EXIST($q(\leq), t$) iff $b \geq BOT(t)(a)$. □

Example 1 *Figure 1 presents an example of dual transformation for a tuple t. Given the half-plane queries $q_1 \equiv y \geq -x - 1$, $q_2 \equiv y \geq x$, $q_3 \equiv y \geq 4.5$, one can see in Figure 1(b) that $-1 < DOWN(t)(-1)$, $4.5 = TUP(t)(0)$ and $DOWN(t)(1) < 0 < UP(t)(1)$. It follows from Proposition 1 that ALL(q_1, t), EXIST(q_2, t) and EXIST(q_3, t) are satisfied. Figure 1(a) confirms these results. Similarly, if we consider the queries $q'_1 \equiv y \leq -x - 1$, $q'_2 \equiv y \leq x$, and $q'_3 \equiv y \leq 4.5$, it follows from Proposition 1 that EXIST(q'_2, t) and ALL(q'_3, t) are satisfied (see Figure 1(a)).* ◇

3 A technique to solve simplified ALL and EXIST problems

As we have seen in the previous section, the dual transformation uniformly reduces both ALL and EXIST selections to a point location problem. For this problem, some efficient in-memory algorithms have been proposed (see [14] for a survey). However, similar algorithms designed for external storage are still far from being adequately efficient [1, 2, 9, 10].

When the angular coefficient of the line associated with the query half-plane belongs to a predefined set S, an optimal dynamic solution to this problem exists [4].[7] The approach is based on the following considerations. For each value $a \in S$, we construct two sets containing the highest (lowest) intersection points of line $x = a$ in $\mathcal{D}$ with all $DOWN(t)'s$ and all $UP(t)'s$, respectively. Each set contains at most N points that can be organized in two lists, denoted by L_{down} and L_{up}, ordered with respect to increasing y values. Given a query tuple $y \ \theta \ ax + b$, by Proposition 1, the position of b in the total order allows to determine the result of the query. Indeed:

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

Thus, to perform selections against a set of tuples, it is sufficient to maintain two ordered sets of values in two B$^+$-trees, B^{up} and B^{down}, for each value in set S.

Given a query $Q(y \ \theta \ ax + b, r)$, such that $Q \in \{$ALL,EXIST$\}$, the search algorithm first selects the corresponding B$^+$-tree associated with value a, 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 is searched in the selected B$^+$-tree and all leaf values are swept in the direction corresponding to the query.[8]

Theorem 1 *Let r be a relation containing N tuples. Let q be a query half-plane. Let T be the cardinality of the set ALL(q, r) (EXIST(q, r)). If the angular coefficient of $p(q)$ is 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(q, r) and EXIST(q, r) selections are performed in $O(\log_B N/B + T/B)$ and updates in $O(k \log_B N/B)$ I/O operations.* □

4 Approximation techniques to solve ALL and EXIST problems

The technique we have proposed in the previous section works only for queries whose angular coefficient belongs to a predefined set. In the following, we propose two solutions overcoming such limitation, based on a filtering/refinement approach. The first technique (denoted by T1) replaces the original query with two new queries whereas the second technique (denoted by T2) replaces the original query with only one new query.

Both techniques use the indexing structures defined in Section 3. The first technique can always be applied,

[6] $DOWN(t)(a)$ ($UP(t)(a)$) denotes the highest (lowest) y-coordinate of the point of $DOWN(t)$ ($UP(t)$) having a as x-coordinate.

[7] This type of selection is typical of VLSI and CAD design.

[8] Another solution to the same problem can be provided by reducing ALL and EXIST selections to the 1-dimensional interval management problem (see [4] for additional details).

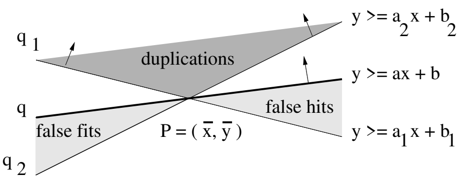

Figure 2. An approximation example

whatever the database is. The second technique can be applied only if the database satisfies some specific requirements. In the following, we denote with S a predefined set of angular coefficients, and assume that B$^+$-trees B^{up} and B^{down} are constructed for each value in S. We consider a query half-plane $q \equiv y \; \theta \; ax + b$, where $a \notin S$.

4.1 Approximating the query with two new queries

From Figure 2, we can see that the original query can be approximated with two new queries (called *app-queries*), with half-planes $q_1 \equiv y \; \theta_1 \; a_1 x + b_1$ and $q_2 \equiv y \; \theta_2 \; a_2 x + b_2$, $a_1, a_2 \in S$, such that the union of q_1 and q_2 covers the original query half-plane. 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 2). Such approximation raises two main issues:

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, 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.

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 down-query $y \geq ax + b$ (similar conditions can be given for up-queries).

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

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

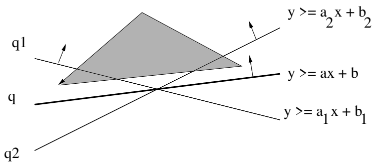

Figure 3. Choice of the type of app-queries for an original ALL query

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 2).

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 (see Figure 3). To preserve correctness, we approximate an ALL query with one EXIST and one ALL app-query. See [6] for additional details.

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.

Theorem 2 *Let r be a relation containing N tuples. Let q be a query half-plane. Let T be the cardinality of the* *set $ALL(q,r)$ ($EXIST(q,r)$). 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 $ALL(q,r)$ and $EXIST(q,r)$ selections are performed in $O(log_B N/B + T_1/B + T_2/B)$ I/O operations, where T_1 and T_2 represent the number of tuples returned by the app-queries generated as above, and updates are performed in $O(klog_B N/B)$ I/O operations.* □

4.2 Approximating the query with a single new query

From Figure 2 we can see that the query can be approximated by using a single new query if it is possible to replace the original query half-plane with a new half-plane such that the part of the original half-plane not covered by the new one does not contain the extension of any tuple. This is possible if we know something about the distribution of the extension of the tuples in the plane. The notion of *direction half-plane* gives this kind of information, specifying that none is present in a given region of space.

Definition 1 *Let r be a relation. A line l is a direction line for r if the extension of r is contained in a single half-plane with respect to l. Such half-plane is called direction half-plane for r.* □

Given a query $Q(q)$, such that $Q \in \{$ALL,EXIST$\}$, if at least one direction half-plane q_1 exists such that $p(q_1)$ and $p(q)$ are not parallel lines, at least one query $Q(q')$ exists approximating $Q(q)$, such that $p(q)$ and $p(q')$ are not parallel lines. Note that if the original query line and the direction line are parallel, the direction line does not give enough information to find a new query approximating the original one, excluding queries whose query half-plane contains the direction half-plane (thus, retrieving all database objects). If $p(q)$ and $p(q_1)$ are not parallel lines, we say that q_1 is *approximating* for q.

Given a query $Q(y\ \theta\ ax + b)$, in order to construct the new query $Q(y\ \theta'\ a'x + b')$, the following choices must be taken.

Choice of the direction half-plane. In order to detect direction lines, a (possibly open) minimum bounding polybox of the relation extension can be maintained. If such a polybox does not exist, the relation does not admit any direction line. Otherwise, the lines on which the edges of the polybox lay represent direction lines for the relation. Of course, direction lines with different angular coefficient may exist. Some heuristics can be applied to choose the direction line generating the lowest number of false hits. See [6] for details.

Choice of the point. Let $y\ \theta_1\ a_1x + b_1$ be the approximating direction half-plane. Since we assume that $a_1 \neq a$, the direction line and the query line $y\ \theta\ ax + b$ intersect. We choose their intersection point to construct the new query. The reason of this choice is due to the fact that for each query half-plane, whose associated line passes for this point, it is immediate to establish if the part of the original query half-plane not covered by the new one contains the extension of some tuple.

Conditions on a, a_1	θ_1	a'	θ'
$a_1 > a$	$\leq$	$max\{a' \mid a' \in S, a' < a\}$ or	$\geq$
		$max\{a' \mid a' \in S, a' \geq a_1\}$	$\leq$
$a_1 < a$	$\leq$	$max\{a' \mid a' \in S, a' > a\}$ or	$\geq$
		$max\{a' \mid a' \in S, a' \leq a_1\}$	$\leq$
$a_1 > a$	$\geq$	$min\{a' \mid a' \in S, a < a' \leq a_1\}$	$\geq$
$a_1 < a$	$\geq$	$max\{a' \mid a' \in S, a_1 \leq a' < a\}$	$\geq$

Table 2. Choice of the app-query half-plane, for an original down-query

Choice of the app-query. Table 2 summarizes the various cases for constructing the app-query half-plane for an original down-query. Similar conditions can be given for an up-query. The type of the app-query coincides with the type of the original one. Note that, if no app-query can be found, T2 cannot be applied.

Theorem 3 *Let S be a set of angular coefficients. Let r be a relation containing N tuples. Let q be a query half-plane. Let T be the cardinality of the set ALL(q,r) (EXIST(q,r)). Assume that there exists at least one approximating direction half-plane for r. If the angular coefficient of $p(q)$ is not contained in S, 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 $O(\log_B N/B + T_1/B)$ I/O operations, where T_1 represents the number of tuples returned by the app-query constructed as above, and updates are performed in $O(k\ \log_B N/B)$ I/O operations.* □

When compared with technique T1, T2 does not generate duplicates, since only one app-query is generated. However, differently from T1, it cannot be always applied. No clear relationship exists between the number of false hits generated by the two techniques. It essentially depends on the choice of point P for the first technique and on the choice of direction lines for the second one.

5 Experimental results

In order to compare the performance of the proposed techniques, we have performed two different groups of experiments, the first comparing techniques T1 and T2, the second comparing techniques T1 and T2 with respect to the R-tree, a well known spatial data structure [11].

In both the considered techniques, each B$^+$-tree is associated with two data files (called UP-file and DOWN-file)

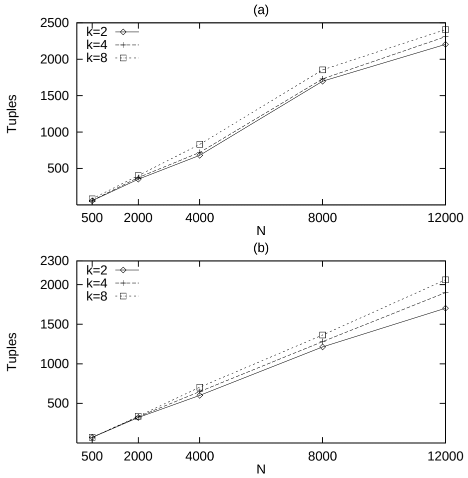

Figure 4. Duplicates generated by T1 (a) for an EXIST selection; (b) for an ALL selection

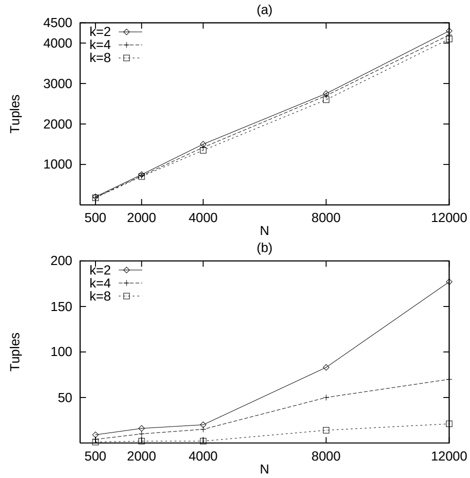

Figure 5. False hits generated for an EXIST selection by (a) T1, (b) T2

containing the dual representation of the tuples, ordered following the ordering induced by the corresponding B$^+$-tree. A similar approach has been taken for implementing refinement in the R-tree. Note that, even if this approach increases the redundancy of the data representation, since tuples are replicated $2k$ times, it improves the query time, since only one leaf node per search is accessed. Moreover, even if this solution could be not feasible from the point of view of the space occupancy, it does not alter the results of the comparison.

The experiments have been performed on a PC Pentium 200, under Windows 95. The page size is 1k. The program has been written in C++. The considered relations contain respectively 500, 2000, 4000, 8000, and 12000 tuples; each tuple contains at most 30 constraints. Both ALL and EXIST selections have been investigated.

5.1 Comparing the performance of T1 and T2

The aim of the experiments is to analyze the trade-off existing between T1 and T2, in order to assess the impact of duplicates and false hits on the search. In doing that, we have assumed that the set S contains angular coefficients of lines dividing the space in $2k$ equal sectors. In the performed experiments we have chosen $k = 2, 4, 8$. We have observed that the trade-off between the techniques does not change by changing the selectivity of the query. For this reason, all results we report here are related to a single query. Moreover, similar results have been obtained for relations containing closed or open objects. Open tuples have been constructed in such a way to guarantee the existence of at least one direction line. Since closed relations will be considered in Subsection 5.2, here we report results obtained for relations containing at least one open tuple.

Duplicates. Figure 4 shows that the number of duplicates increases for increasing values of k. Indeed, for higher values of k, the common area of the two app-query half-planes increases. Therefore, more tuples are returned twice.

False hits. Figure 5 shows that the number of false hits decreases for increasing values of k, for an EXIST selection. Similar results have been obtained for the ALL selections. This behavior is reasonable since higher values for k correspond to smaller false hits areas. Note that T2 generates the lowest number of false hits. This is mainly due to the particular type of relations used in the experiments.

Page accesses. Similar results have been obtained for all k-values. Figure 6 reports results obtained for $k = 2$. It can be observed that T2 performs better than T1. This is mainly due to the fact that T2 does not generate duplicates.

5.2 Comparing the performance of T1, T2, and R-trees

In order to establish the practical applicability of the proposed techniques, we have compared their performance with respect to the performance of the R-tree [11], a well known spatial data structure for closed objects. The R-tree can be used to answer EXIST and ALL queries. However, in order to safely execute an ALL selection with respect to a query half-plane, the selection has to be replaced by the corresponding EXIST selection; the result has then to be refined with respect to the original ALL query.

Several experiments have been performed, by varying the following parameters:

- *The average size of the considered objects.* Three different groups of objects have been considered: *large*

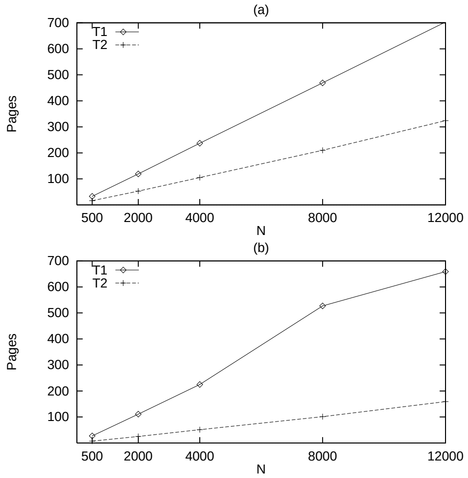

Figure 6. Page accesses for $k = 2$ and (a) an EXIST selection, (b) an ALL selection

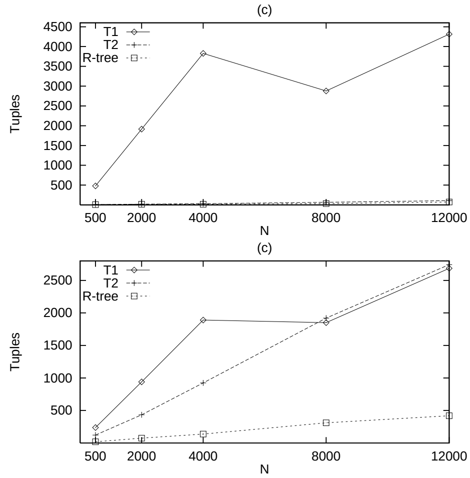

Figure 7. False hits for an EXIST selection and (a) $s < 10\%$; (b) $s \approx 40\%$

objects, i.e., objects intersecting almost all other objects; *medium* objects, i.e., objects whose area does not exceed half the area of the bounding rectangle containing all stored ones; *small* objects, i.e., objects with a very small area with respect to the bounding rectangle containing all stored ones. All objects are uniformly distributed in the space. 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. In this paper, due to space constraints, we only report results obtained for small rectangles.

- *The selectivity of the considered queries.* In comparing T1 and T2 with R-trees, selectivity (hereafter denoted by s) is very important since different selectivities correspond to a different number of internal tree nodes accessed in the R-tree. Here, we report results obtained with $s < 10\%$ and $s \approx 40\%$.

In performing the experiments, we have taken $k = 2$. This assumption allows us to compare R-trees with respect to the proposed techniques in the case when they have the worst performance (see Subsection 5.1).

In the following, due to space constraints, we mainly report the results we have obtained by considering EXIST selections. See [6] for additional details. The technique supported by the R-tree data structure is denoted by R.

False hits. From the performed experiments, it follows that R almost always generates the lowest number of false hits. For selectivity very low ($< 10\%$), the number of false hits generated by T2 is very close to the number of false hits generated by R. Often T2 generates less false hits than T1, but this mainly depends on the chosen relation and on the choice of point P. These results can be observed from Figure 7. From the same figure, we can see that, by increasing the selectivity, the number of false hits generated by T1 decreases whereas the number of false hits generated by T2 and R increases. In R, the number of false hits increases because, by augmenting the selectivity, the number of tree paths to be searched increases. In T1, the number of false hits decreases since the false hits area decreases whereas in T2 increases, thus increasing the number of generated false hits. From additional experiments, we have also observed that R generates the lowest number of false hits when rectangles are small. These considerations point out an important difference between R-trees and the proposed data structures: the performance of a search based on R-trees depends on the query selectivity whereas the performance of T1 and T2 depends on the size of the false hits area generated by the approximation. Thus, by choosing a good approximation, similar performance can be obtained when executing queries with different selectivity. Similar results have been obtained for ALL selections.

Page accesses. Different results have been obtained by considering the number of page accesses. In this case, T2 almost always performs better than R. This is in contrast with the result deriving from the analysis of false hits and is mainly due to the number of tree paths that have to be analyzed in the R-tree. Indeed, in T2, always a single path of a B$^+$-tree has to be analyzed. In the performed experiments, this corresponds to at most 3 page accesses. On the other hand, each single query may require the analysis of several paths in the R-tree, depending on the query selectivity. From the experimental results, it follows that the number of additional page accesses required to search the R-tree is higher than the number of additional pages that have to be analyzed in T2 for the additional false hits. These results can be observed

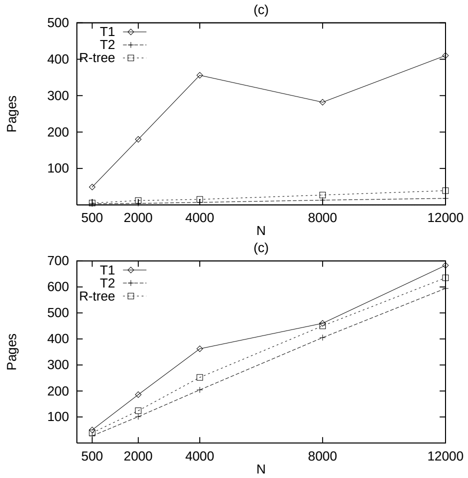

Figure 8. Page accesses for an EXIST selection and (a) $s \leq 10\%$**; (b)** $s \approx 40\%$

from Figure 8. The reported results show that, similarly to the analysis of false hits, by augmenting the selectivity, the number of pages accessed by T2 and R increases. However, differently from the result obtained by the analysis of false hits, the number of pages accessed by T1 increases by increasing the selectivity. This is mainly due to the fact that T1, besides the generation of false hits, also generates duplicates. This aspect, together with the fact that the number of tuples belonging to the result increases by increasing the selectivity, increments the number of page accesses. A similar situation arises when ALL selections are considered. However, the performance of the R-tree is much worst for ALL selections than for EXIST selections, with the same selectivity. This is due to the fact that the search in the tree for an ALL selection coincides with a search in the tree for a corresponding EXIST selection and therefore the degree of filtering is lower.

6 Conclusions and future work

The paper has proposed two new approximation techniques to solve ALL and EXIST selections in two-dimensional linear constraint databases. The proposed solutions are based on a dual representation, first presented in [8, 10]. We have also experimentally compared the proposed techniques with R-trees. Future work includes: the definition of new approximation techniques, not generating duplicates and reducing the space overhead; additional experimental work, in order to compare the proposed techniques with R^+-trees; the extension of the proposed techniques to a d-dimensional space, $d > 2$.

References

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] B. Catania. *Constraint Databases: Data Models and Architectural Aspects*. PhD thesis, University of Milano, Italy, 1998.

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

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

[9] 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.

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

[11] 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.

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

[13] 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.

[14] P.K. Agarwal and J. Erickson. Geometric Range Searching and its Relatives. In *Discrete and Computational Geometry: Ten Years Later*. American Mathematical Society, Providence, 1998.

[15] F. Preparata and M. Shamos. *Computational Geometry - an Introduction*. Springer Verlag, 1985.

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

[17] 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.

[18] 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.