Using Constraint Programming in Selection Operators for Constraint Databases

María Teresa Gómez-López *, Rafael M. Gasca

Department of Languages and Computer Systems, University of Seville, Spain

A B S T R A C T

Constraint Databases represent complex data by means of formulas described by constraints (equations, inequations or Boolean combinations of both). Commercial database management systems allow the storage and efficient retrieval of classic data, but for complex data a made-to-measure solution combined with expert systems for each type of problem are necessary. Therefore, in the same way as commercial solutions of relational databases permit storing and querying classic data, we propose an extension of the Selection Operator for complex data stored, and an extension of SQL language for the case where both classic and constraint data need to be managed. This extension shields the user from unnecessary details on how the information is stored and how the queries are evaluated, thereby enlarging the capacity of expressiveness for any commercial database management system. In order to minimize the selection time, a set of strategies have been proposed, which combine the advantages of relational algebra and con-straint data representation.

Keywords:
Complex data
Optimal query evaluation
Selection Operator
Constraint Databases
Constraint Programming

1. Introduction

Databases are used for the management of information in software applications. However there are certain types of data that cannot be represented as classic data and treated with commercial solutions, such as software mining, web analytics, medicine, biology and chemistry data (Poelmans, Ignatov, Kuznetsov, & Dedene, 2013a; Poelmans, Kuznetsov, Ignatov, & Dedene, 2013b). Constraint Databases represent complex data by means of formulas described by constraints (equations, inequations or Boolean combinations of both). The problem remains that no commercial database exists that permits constraints to be handled in the same terms as classic data, and which shields the user from how the information is stored and how the queries are evaluated. It implies that, for each type of problem, a made-to-measure solution combined with expert systems are necessary. The use of the constraint data type to represent these different types of problems permits the abstraction to use the data in the same sense as in the relational databases, where the same type of data can be used for different semantics.

The main contributions of this paper can be summarized as follows: (1) An extension of SQL language to evaluate the Selection Operator for complex data; (2) The proposal for a set of strategies to minimize the evaluation time; (3) A library to extend commercial databases for constraint data and Selection Operator; (4) It is applied to a project for the Human Accident Risk Management in Construction.

The industrial development of the proposal has been elaborated in a real project called Victor R&D Project. The Victor Project (Visual Interface ConTrol Object Rules) (Victor-Project, 2013) is a case study where classic and complex data are managed at the same time in an application, and the efficiency of the data selection is essential for security reasons. This project focuses on the Human Accident Risk Management in Construction, where it is possible to model static objects at design time (video cameras, ditches, cranes, machinery zones, ...), dynamic objects whose position is sent from the sensors at runtime (persons, machinery, ...), and rules that describe the safe or dangerous relation between the objects. The various objects have locations that can be described by means of points, lines, polygons, or irregular zones, and for different floors, in general by means of constraints (Fig. 1). The data retrieval over this data implies the evaluation of queries such as: Which machines are in a zone of security on floor three? or Is there anybody with a level of risk higher than 8 from a ditch? In order to describe the data model of the project, the level of floor can be represented by means of an integer, and the type of zone or the type of

* Corresponding author. Tel.: +34 954553871; fax: +34 954557139.
E-mail addresses: maytegomez@us.es (M.T. Gómez-López), gasca@us.es (R.M. Gasca).
URLs: http://www.lsi.us.es/~mayte (M.T. Gómez-López), http://www.lsi.us.es/~gasca (R.M. Gasca).

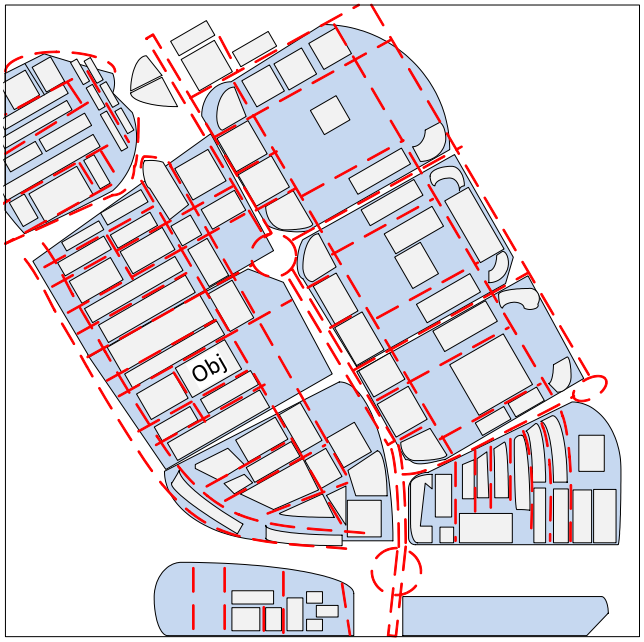

Fig. 1. Example of objects of risk management.

static element can be stored by means of an enumerate type, but the problem remains of how to represent and to ascertain whether a piece of machinery is inside or outside of a zone, or how to represent the risk between two objects using only operations over the database. The development of this type of query implies a made-to-measure implementation depending on the problem. However, would it be possible to extend the Selection Operator to query complex types of data represented by means of constraint, in the same sense as classic data is treated in relational databases? Optimal data selection is essential since there are many objects, and a delay in identifying potential accidents can be very negative for the safety of the personnel. For this reason, the main objective of this paper is to improve the constraint data retrieval, by combining the advantages of the commercial relational databases in order to delegate the evaluation of the classic data to the database management.

This paper is organized as follows: Section 2 presents how complex data represented by constraints can be stored using Constraint Databases. Section 3 extends the syntax and the semantics of the Selection Operator of relational algebra for constraint data. Sections 4 and 5 explain how the queries can be evaluated by describing the implementation details based on Constraint Satisfaction Problems, and the strategies to improve the evaluation time. Section 6 presents several examples of queries and their average evaluation times, and shows how the use of our solution improves the computation time for various selections over constraints. Section 7 reviews the most relevant proposals in this area. Finally, conclusions are drawn and future work is proposed.

2. Constraint Databases background

Constraints have already been related to databases by means of the so-called Constraint Databases (CDBs) (Revesz, 2001). Constraint Databases were introduced in 1990 in a paper by Kanellakis and Revesz (1990), creating a new line of research (Revesz, 1995Revesz, 1998).

Although there are several proposals related to CDBs, the definitions of CDBs and how to store the constraints used in this paper is the proposal called LORCDB (Gómez-López, Ceballos, Gasca, & Valle, 2009), which is based on Chapter 2 of Revesz (2001) with certain variants. In Section 7 the different solutions found in the literature are analysed and why the LORCDB solution is used to extend the Selection Operator in this paper. The necessary definitions are included in this work to facilitate the understanding of the paper.

A *Constraint* can be expressed by means of a Boolean combination of equations and inequations that follows the metamodel presented in Gómez-López, Gasca, and Reina-Quintero (2011). A *constraint* is a Boolean combination (not, and, or) of comparison ($<, \leqslant, \geqslant, \ldots$) among equations ($+, *, -, /$). For example: {a>b AND NOT c + 1 = b}

When constraint data need to be stored in a database, classic attributes and constraint attributes can be combined in the same relation, where the tuples of the relation are called constraint *k*-tuples:

- A *constraint k-tuple* with the variables $x_1, \ldots, x_k$ is a finite conjunction $\varphi_1 \wedge \ldots \wedge \varphi_N$ where each φ_i, for $1 \leqslant i \leqslant N$, is either a constraint, such that $\{x_j$=*Constant*$\}$, where $x_j \in \{x_1, \ldots, x_k\}$, called a *Classic Attribute*, or an Ω-constraint over the variables $x_1, \ldots, x_k$ which do not correspond to a classic attribute, and is called a *constraint Attribute*.

When a set of constraint k-tuples are joined in a relation, it is called a constraint relation:

- A *constraint relation* is defined as a finite set of *Classic Attributes* and *constraint Attributes*. A *constraint relation of arity k*, is a finite set $r = \{\psi_1, \ldots, \psi_M\}$, where each ψ_j for $1 \leqslant j \leqslant M$ is a constraint *k*-tuple over $\{x_1, \ldots, x_k\}$. The corresponding formula is the disjunction $\psi_1 \vee \ldots \vee \psi_M$, such that $\psi_j = \varphi_1 \wedge \ldots \wedge \varphi_N$ for each φ_i is a *constraint k-tuple*, where $1 \leqslant i \leqslant N$. If in each $\psi_j \in r$ there is a φ_i such that $\{x$ = *Constant*$\}$, where x is the same variable in all φ_i belonging to different ψ_j, and x does not appear in the rest of the φ_i of the same ψ_j, then the x variable is a **classic attribute**, while the rest of the variables belong to **constraint attributes**.
  A relation has classic attributes if and only if:
  φ_{ij} is a $\varphi_i \in \varphi_1 \wedge \ldots \wedge \varphi_N$ and $\psi_j \in \psi_1 \vee \ldots \vee \psi_M$, such that $\psi_j = \varphi_{1j} \wedge \ldots \wedge \varphi_{Nj}$, and therefore a constraint relation will have a classic attribute (x) if:
  $\exists \varphi_{ij} \bullet \forall j \in 1..M \mid i \in 1..N, \{\varphi_{ij} \equiv x = c_j\} \wedge$
  $\forall t \in 1..N \wedge t \neq i \wedge \varphi_{tj}(x_1, \ldots, x_k) \wedge x \notin \{x_1, \ldots, x_k\}$

where c_j is a constant, M the number of tuples, and N the number of attributes (columns).

This implies that if an equality relation exists between a variable and a constant (the same variable in all tuples) in all constraint *k*-tuples, and that if this variable does not appear in another constraint attribute, then this variable is a classic attribute, since it follows relational algebra.

An example is presented in Fig. 2, where the constraint relation is composed of one constraint attribute and two classic attributes. In the example, there are two variables (x and y) that appear in all the tuples which have an equality relation with a constant, and these variables cannot appear in the rest of the attributes.

Therefore, a *Constraint Database* is a finite collection of constraint relations composed of **Classic and constraint attributes**. From the previous definitions, it is clearly necessary to extend the types of attributes for CDBs. The definitions of three different types of attributes are used:

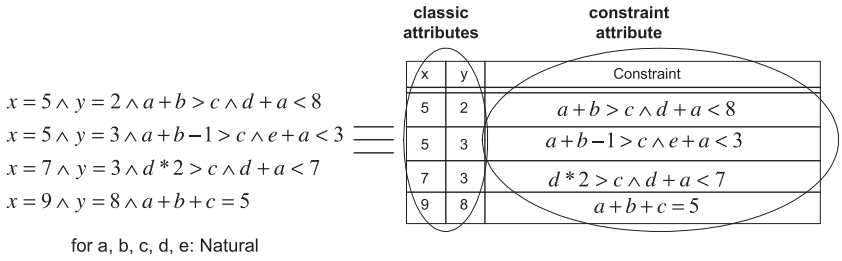

Fig. 2. Example of constraint k-tuples and constraint relations.

- **Classic attribute** ($\mathtt{at}_i$)**:** at_i belongs to the n-tuples of a relation ($1 \leqslant i \leqslant n$), where at_i is of a type permitted by standard SQL, for example *Integer*, *String*, *Date*, ... This type of attribute can be represented as a constraint, such as {at_i = constant}, since relational databases only permit attributes with atomic values. These attributes can be used as a primary or foreign key.
- **Constraint attribute** ($\mathtt{at}_i^c$)**:** at_i^c belongs to the n-tuples of a relation where at_i^c is of *Constraint* type defined as a *constraint k-tuple* with the variables $v_1 \ldots v_k$. A constraint attribute represents a relation between the set of variables $v_1 \ldots v_k$. Therefore, for our proposal, a constraint relation is formed by classic and constraint attributes.
- **Constraint-variable attribute** ($\mathtt{at}_i^c.\mathtt{v}_j$)**:** $\mathtt{v}_j$ is a variable which belongs to a constraint attribute at_i^c. This attribute is represented by:
  `<Constraint_column_name>.<Variable_name>`

where `<Variable_name>` is not a classic attribute, but it is a variable of a constraint stored as a value inside a column. A column of the database (constraint attribute) can have several constraints, and these constraints can be defined over different variables.

Derived form theses definitions, The storage proposal in Gómez-López et al. (2009) presents a framework called LORCDB (Labelled Object–Relational Constraint Database). LORCDB is focused on storing the constraints as objects indexed by the variables contained within, the minimum and maximum value of each variable for every constraint, what is the box consistency for each variable (the minimum-bounding hyper-rectangle) (Granvilliers, Goualard, & Benhamou, 1999).

By storing the maximum and minimum values, the query evaluation becomes more efficient, as explained in the following sections.

3. Extending the syntax and semantics of the Selection Operator for constraint data

One of the main objectives of this paper is to define a language to describe the constraint data for the selection of data in a database, independent of the semantics of the problem. Since SQL is the standard language for relational databases, we propose an extension to the meaning of selection for constraint data. For the extension, it is necessary to propose: a new syntax; the semantics of the possible comparators; and a way of evaluating the queries in order to optimize the data selection. Examples where a Selection Operator can be used to select data include:

- Is any security zone being crossed by any machine?
- What degree of risk does the ditch $(x-5)^2 + (y-12)^2 \leqslant 36 \wedge 2 <= x \wedge x <= 8$ pose for a particular person?
- What objects are to the north of the person located at the point $(6, 8)$?

In general, the Selection Operator has to return the tuples that satisfy a condition, which can relate an attribute either with a constant or with attributes of different relations. An example of a selection in SQL is:

`Select R.* from R, R' where R.`At_3 θ `R' .`At_2

The main difference in the syntax, when constraint data is added, is that the attributes can be constraints, and hence the comparison (θ) must allow the comparison between constraints. One of the goals of this paper is to define what "to compare constraints" means, what types of comparison can be defined between constraints, how we can evaluate the queries, and how the efficiency of query evaluation can be improved. In this paper we propose various strategies to improve the evaluation, such as the box consistency of constraints, relational algebra, and Constraint Satisfaction Problems.

3.1. Syntax of the Selection Operator

The syntax of the query with a selection of tuples in standard SQL is:

`SELECT <list_of_attributes (t1) [,list_of_attributes (t2), ...]>`

`FROM <list_of_tables (t1, t2, ...)> WHERE predicate`

The *predicate* is a condition where the attributes of tables involved can participate, where a Boolean expression is composed of logic operators $\chi = \{\wedge, \vee\}$ and comparison operators $\theta = \{<, \leqslant, >, \geqslant, =, <>\}$ with the form:

$a_1 \theta c_1 \ \chi \ a_2 \theta c_2 \ \chi \cdots \chi \ a_n \ \theta \ c_n$

where a_i represents an attribute (one of the three proposed for CDBs) and where c_i is another attribute or a constant of the same domain as that of a_i.

The differences added for the extension of SQL for constraints are: the type of attributes involved, which can be constraint attributes; and the predicate to define the condition. The parameter `<list_of_attributes (ti)>` represents various types of projections over the attributes of table t_i. These attributes can be classic attributes, constraint attributes, or constraint-variable attributes. The predicate that we propose extends the standard, adding to the classic attributes the comparison among constraint attributes, constraint-variable attributes, and constraint attributes for a set of variables (at, at_c, at_c.variable). The operators to compare classic attributes ($<, \leqslant, >, \geqslant, =, <>$) are also used to compare constraint-variable attributes, but the operators to compare constraint attributes are extended to compare the constraints ($<, \leqslant, >, \geqslant, =, <>, \&, \subset, \subseteq, \supset, \supseteq$). When only a number of the variables of the constraints are to be taken into account in the selection process, the syntax permits either the specification of these variables with the token `FOR`, or a combination of this comparison with the equality of other variables. A detailed clarification of the following syntax is given in the subsection below:

```
predicate := condition
    | condition ['AND' | 'OR'] predicate;
condition := at COMPARATOR [at | constant]
  | at_c CONSTRAINTCOMP [at_c | constraint]
  | at_c CONSTRAINTCOMP '('[at_c | constraint] 'FOR'
'{'listOfVariables'}'
    ['AND' listAtVarEqual]')'
  | at_c'.'variable COMPARATOR [at_c'.'variable |
constant];
COMPARATOR := '<' | '⩽' | '>' | '⩾' | '=' | '<>';
CONSTRAINTCOMP := '<' | '⩽' | '>' | '⩾' | '=' | '<>' |
'&' | '⊂' | '⊆'
  | '⊃' | '⊇';
listOfVariables := Variable
  | Variable ',' listOfVariables;
listAtVarEqual := at_c'.'Variable '='
at_c'.'Variable
  | at_c'.'Variable '=' at_c'.'Variable 'AND'
listAtVarEqual;
```

3.2. Semantics of the Selection Operator for constraint attributes

In the same way as the comparison operators have been modified, the semantics has to be adapted. In this work, the comparison defined between constraints is related to the comparison between the extensional values that the constraints involved represent in an intensional way. As stated earlier, the constraint data is an intensional way to represent a set of extensional data, therefore their comparison operators must compare all the extensional data that they represent. Some of the operators used are those defined in Chapter 2 of Marriott and Stuckey (1998) that describe the relation between the extensional values of the variables represented by means of constraints in a similar way to operations between sets. Other papers, such as Lee, Unger, Zheng, and Lee (2011), also use the comparison operators for the containment scope of mobiles in covered zones. We propose the comparison which ascertains whether the values that represent a constraint are included in the values of another constraint ($\subset$, $\subseteq$, $\supset$, $\supseteq$), or that both sets of values of the variables have at least one solution in common ($\&$). Moreover, the semantics of the classic operator for the comparison in relational algebra have also been added ($\theta = (<, \leqslant, >, \geqslant, =, <>)$). The semantics defined in this paper for these operators describes whether the values that represent a constraint are greater than, smaller than, equal to or different from the values of another constraint. The typical topological relations (disjoint, meet, overlap, ...) can be performed with the comparison operators that we propose. The advantage of incorporating these operators, however, is that they can be used in any type of problem or selection, by permitting the use of the same operators in a general purpose database, and not only for spatio-temporal data.

We also find it interesting to compare a set of the variables involved only in the constraints, for this reason we have included a special syntax: `'('[at_c | constraint] 'FOR' '{'listOfVariables'}'')'`. This sentence allows the definition of the set of variables that we want to include in the comparison.

Before explaining how the evaluation of the comparison is performed, it is necessary to clarify that if the involved constraints in the comparison are not defined over the same variables, then the comparison has no sense. For this reason, we define a new operation over the variables that describes the syntactic equality between variables of different constraints:

Definition 3.1. Syntactic Equality between variables of constraints: Let C_a and C_b be two constraints where $A = \{a_1, a_2, \ldots, a_n\}$ represents the variables of C_a, and $B = \{b_1, b_2, \ldots, b_m\}$ represents the variables of C_b. If $a_i \in A$, and $b_j \in B$, then $a_i \equiv_S b_j$ is true if both variables are syntactically equal, which means that they have the same name. If two variables of different constraints are syntactically equal, it does not imply that they are the same variable and represent the same extensional values in both constraints. For example, if two separate objects are defined in a zone of the space, both objects are described by the same variables (x, y). However there is no solution for both objects for the same values of x and y, although it is possible to define a comparison between them, for example if one object is higher than the other. This means that although the variables of two constraints of different tuples are syntactically equal, they are not semantically equal because they represent symbolically different extensive values.

4. Evaluation of Selection Operator for data selection

In order to compare the values represented by means of constraints, and to evaluate the queries over the possible constraint attributes, we propose the use of Constraint Satisfaction Problems because their resolution is more efficient and general (for any number of variables) than the computation geometry algorithm used in other proposals. Depending on the related constraints in the evaluation, and the comparator of the Selection Operator, various models are created to evaluate the query. Moreover, classical mathematics models should not be applied to solve geographical problems, then new ways to represent and solve them are necessary (Malpica, Alonso, & Sanz, 2007). In order to understand the models, it is necessary to clarify what a Constraint Satisfaction Problem is, to determine whether a comparison between two constraints is *true* or *false*, and to determine whether the tuple with the constraint will form part of the output constraint relation.

Constraint Satisfaction Problems (CSPs) represent a reasoning framework consisting of variables, domains and constraints. Formally, it is defined as a triple $\langle X, D, C\rangle$ where $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables, $D = \{d(x_1), d(x_2), \ldots, d(x_n)\}$ is a set of domains of the values of the variables, and $C = \{C_1, C_2, \ldots, C_m\}$ is a set of constraints. Each constraint C_i is defined as a relation R on a subset of variables $V = \{x_i, x_j, \ldots, x_k\}$, called the *constraint scope*. The relation R may be represented as a subset of the Cartesian product $d(x_i) \times d(x_j) \times \ldots \times d(x_k)$. A constraint $C_i = (V_i, R_i)$ specifies the possible values of the variables in V simultaneously in order to satisfy R. Let $V_k = \{x_{k_1}, x_{k_2}, \ldots, x_{k_l}\}$ be a subset of X, and an l-tuple $(x_{k_1}, x_{k_2}, \ldots, x_{k_l})$ from $d(x_{k_1}), d(x_{k_2}), \ldots, d(x_{k_l})$ can therefore be called an *instantiation* of the variables in V_k. An instantiation is a solution only if it satisfies the constraints C.

In order to solve a CSP, a combination of search and consistency techniques is commonly used (Dechter, 2003). The consistency techniques remove inconsistent values from the domains of the variables during or before the search. Several local consistency and optimization techniques have been proposed as ways of improving the efficiency of search algorithms.

The technique for comparison of the constraints proposed in this paper uses model-driven evaluation of CSPs. This implies building a CSP to ascertain whether each constraint stored in the relation satisfies the condition defined in the query, which determines whether each tuple of the input constraint relation will form part of the output constraint relation, and hence a CSP is modelled and solved. The inclusion of each tuple within the group of those that satisfy the condition, and therefore its inclusion in the output relation, depends on the condition and on the existence of a solution for the CSP. The comparison between two constraints is defined if there is a syntactic equality between the variables of the constraints (Definition 3.1) and if the variables are defined over the same domain. Otherwise, the result of the evaluation will be *false*. Given that $a_1 \ldots a_n$ represent the variables of constraint C_a,

and $b_1 \ldots b_m$ represent the variables of C_b, then the created models for the comparison operators are:

- $C_a < C_b$ is true if, for all $a_i \mid a_i \equiv_S b_j$, the maximum value of the solutions of a_i always remains smaller than the minimum value of b_j, and if for all the variables of C_a, there is a syntactic equality in C_b and vice versa. This definition can be extended to $C_a > C_b$. In order to determine whether $C_a < C_b$, a CSP is built where all the variables a_i of C_a, that also participate in C_b (which means that $a_i \equiv_S b_j$), are renamed as $a_i\prime$, thereby building a new constraint $C_a\prime$. This creation is necessary since these two constraints have to be able to assume different solutions in order to compare the values of the solutions. If a CSP is built with the variables both syntactically and semantically equal, then the solutions obtained are the values of a_i where both constraints are satisfiable. In order to ascertain whether all the solutions of C_a for the variables $a_i \equiv_S b_j$ are smaller than the solutions of C_b, a created CSP will determine whether there is a solution where this does not happen. If a solution is found, then the output of the evaluation for the predicate $C_a < C_b$ is *false*, and *true* otherwise. This means that the formula:
  $\forall a_i \in A,\ \forall b_j \in B \mid a_i \equiv_S b_j \Rightarrow (a_i < b_j)$ is transformed into:
  $\neg(\exists\ a_i\ \in A,\ \exists b_j \in B | a_i \equiv_S b_j \Rightarrow a_i > b_j \vee a_i = b_j)$.
  This transformation obviates the study of the entire search space of the solutions, since it is only necessary to find a value where $a_i \geqslant b_j$ is true. A CSP is built as follows:

$$A' = \{a'_1, a'_2, \ldots, a'_n\}$$
$$B = \{b_1, b_2, \ldots, b_n\}$$
$$C'_a \wedge C_b$$
$$\forall a_i \in A\ \forall b_j \in B \mid a_i \equiv_S b_j \Rightarrow \texttt{add}\ \{(a_i\prime > b_j \vee a_i\prime = b_j)\}$$

  If there exists a solution for the CSP, then $C_a < C_b$ for these tuples returns *false*, since it means that not all the values of the variables of C_a are smaller than those in constraint C_b, since there exists a value that satisfies the opposite condition. If no solution is found for the CSP, then the comparison $C_a < C_b$ returns *true*. An example is shown in Fig. 3, where $C_a < C_b$ is true.
  This type of selection can be used to query spatial objects, for example to ascertain what objects are to the southwest of another object.
- $C_a \leqslant C_b$ is true if, for all the variables $a_i \equiv_S b_j$, the maximum value of a_i is always smaller than or equal to the minimum values of b_j. This definition can be extended to $C_a \geqslant C_b$. The construction of this CSP is as follows:

$$A' = \{a'_1, a'_2, \ldots, a'_n\}$$
$$B = \{b_1, b_2, \ldots, b_n\}$$
$$C'_a \wedge C_b$$
$$\forall a_i \in A\ \forall b_j \in B \mid a_i \equiv_S b_j \Rightarrow \texttt{add}\ \{(a_i\prime > b_j)\}$$

  If there is a solution for this CSP, then $C_a \leqslant C_b$ returns *false*, and *true* otherwise.
- $C_a = C_b$ is true if all the solutions of C_a are also solutions of C_b and vice versa. In this case, it is possible that two syntactically different constraints can be semantically equal. For this comparison, the variables are not renamed for the CSP, since it is necessary to know whether the solutions of C_a are also solutions of C_b. The CSP built in this case is:

$$A = \{a_1, a_2, \ldots, a_n\}$$
$$B = \{b_1, b_2, \ldots, b_n\}$$
$$(\neg C_a \wedge C_b) \vee (C_a \wedge \neg C_b)$$

  If there is a solution for the CSP, then $C_a = C_b$ returns *false*, and *true* otherwise. The operation <>, which describes the inequality of two constraints, can be evaluated with the same construction of the CSP used for the = operator. The main difference when using this operator is that if there is a solution for the CSP, the comparison returns *true*, and *false* otherwise.
- $C_a \ \& \ C_b$ is true if there is a solution that satisfied both constraints, where all the variables are syntactically equal. In this case, the model of the CSP is:

$$A = \{a_1, a_2, \ldots, a_n\}$$
$$C_a \wedge C_b$$

  If a solution is found for this CSP, the operation $C_a \ \& \ C_b$ returns *true*, and *false* if no solution is found.
- $C_a \subseteq C_b$ is true whether all the solutions of C_a are also solutions of C_b. In order to analyse the inclusion operator in constraints, both constraints have to be defined over the same variables. Therefore, since C_a and C_b are two constraints where $A = \{a_1, a_2, \ldots, a_n\}$ are the variables of C_a and C_b, then $C_a \subseteq C_b$ is equal to the implication $(C_a \rightarrow C_b)$ (Marriott & Stuckey, 1998). This comparison determines if all the solutions of C_a are also solutions of C_b, although it is possible that C_b has solutions that do not belong to C_a.
  In order to avoid having to analyse all the solutions of C_a by checking whether they are solutions of C_b, the CSP is modelled to search for solutions where the constraint is not satisfiable. The evaluation of the conditional predicate $C_a \subseteq C_b$ therefore corresponds to the formula:
  $\neg(\exists_{a_i \in A}(C_a \wedge \neg C_b))$

  and the model of the CSP is:

$$A = \{a_1, a_2, \ldots, a_n\}$$
$$C_a \wedge \neg C_b$$

  If any solution is found for the CSP, the evaluation returns *false*, and *true* if no solution is found. There is an example shown in Fig. 4 where the constraint C_a is included in the constraint C_b (all the solutions of C_a are solutions of C_b). This definition can be extensible for $C_a \supseteq C_b$.
- $C_a \subset C_b$ returns *true* if all the solutions of C_a are also solutions of C_b, and C_b has at least one solution which does not belong to

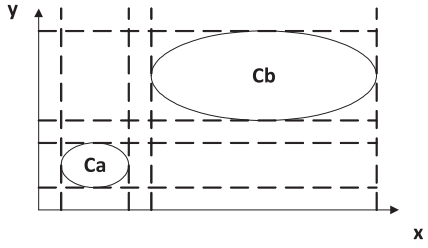

Fig. 3. Example where $C_a < C_b$.

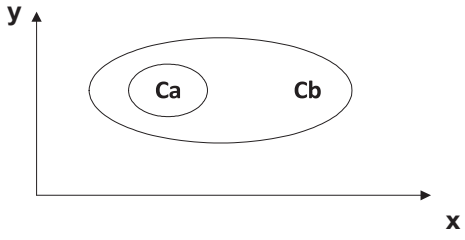

Fig. 4. Example of $C_a \subseteq C_b$.

C_a. As for the previous operator, both constraints have to be defined over the same variables.
In this case, one CSP is modelled in order to determine whether $C_a \subseteq C_b$, and another CSP is modelled to determine whether there is a solution for C_b that does not belong to C_a:

$A = \{a_1, a_2, \ldots, a_n\}$
$C_b \wedge \neg C_a$

If there is a solution for the CSP, then $C_a \subset C_b$ returns *true*, and *false* otherwise. This definition can be extended to $C_a \supset C_b$.

The operators above can be modified to compare only one set of the variables, defined syntactically with the word `FOR`, and to establish equality relations between certain variables:

- $C_a < (C_b$ FOR $\{b_d, \ldots, b_h\})$ is a variant of the comparison '<' which returns *true* if for all the variables $a_i \equiv_S b_j$ such that $b_j \in \{b_d, \ldots, b_h\}$, the maximum value of a_i is smaller than the minimum value of b_j. This comparison carries out a projection over those variables that belong to the predicate $\{b_d, \ldots, b_h\}$. This assumes that the variables of C_a that do not belong to $\{b_d, \ldots, b_h\}$ hold no significance for this operation. In this case, the constraints C_a and C_b do not require all the variables to be syntactically equal, and the comparisons between the variables of the predicate are included in the CSP. The model of the CSP has the form:

$A' = \{a'_1, a_2', \ldots, a'_n\}$
$B = \{b_1, b_2, \ldots, b_n\}$
$C'_a \wedge C_b$
$\forall a_i \in A\ \forall b_j \in \{b_d, \ldots, b_h\} \mid a_i \equiv_S b_j \Rightarrow \texttt{add}\ \{(a'_i > b_j \vee a'_i = b_j)\}$

If there is any solution for the CSP, the comparison returns *false*, and *true* otherwise. An example is presented in Fig. 5a where the selection is over the variable y, and the predicate is $C_a < (C_b$ FOR $\{y\})$. This predicate queries whether all the values of the variable y in C_a are smaller than the solutions for this variable in C_b, where the values of the variables $\{x, z\}$ are not important. However, for the predicate $C_a < (C_b$ FOR $\{x\})$ and the same example, the evaluation output is *false*. This definition can be extended to the operations $\leqslant, >, \&$ and $\geqslant$.

An easy way to illustrate this type of selection is in an application which refers to objects in space, for example, in the determination of whether there is any object physically located above another.

A further variant of this type of comparison for variables is illustrated in ascertaining whether the solutions of a constraint are smaller than the solutions of another constraint for a fixed group of variables. In Fig. 5b an example is shown, in which, for the values of x where C_a and C_b are satisfiable, the value of y to satisfy C_a is smaller than the value of y to make C_b satisfiable. The conditional predicate for this example is:

$C_a < (C_b$ `FOR` $\{y\}$ `AND` $C_a.x = C_b.x)$.
It assumes that the general syntax to fix the semantic equality is:

$C_a < (C_b$ `FOR` $\{b_d, \ldots, b_h\}$ `AND` $C_a.v_1 = C_b.v_1$ `AND` $\ldots$ `AND` $C_a.v_i = C_b.v_i$ `AND` $\ldots$ `AND` $C_a.v_k = C_b.v_{k)}$.

where $v_i \notin \{b_d, \ldots, b_h\}$

and means that the model of the CSP is:

$A' = \{a'_1, a_2', \ldots, a_n\prime\}$
$B = \{b_1, b_2, \ldots, b_n\}$
$C'_a \wedge C_b$
$\forall a_i \in A\ \forall b_j \in \{b_d, \ldots, b_h\} \mid a_i \equiv_S b_j \Rightarrow \texttt{add}\ \{(a'_i > b_j \vee a_i\prime = b_j)\}$
$\forall v \in \{v_1, \ldots, v_k\} \mid v \equiv_S a_i\prime \wedge v \equiv_S b_j \Rightarrow \texttt{add}\ \{a_i\prime = b_j\}$

If the CSP finds a solution, then the comparison will be false, and true if no solution is found.

The remaining operators can be extended for the use of `FOR` in a similar way, although $\subset, \subseteq, \supset, \supseteq, \&, =$ and $<>$ cannot be extended to an equality relation between variables since these operators define a relation where the variables of these two constraints are equal, and hence stating that the variables are equal would be redundant.

5. Strategies for improving the selection of constraint data

In order to define a methodology to obtain an efficient evaluation of the Selection Operator for the data retrieval, a strategy inspired by spatial databases has been designed to handle each part (conditions) of the predicate in a specified order. The evaluation for the selection consists of obtaining a horizontal subset of tuples for an input relation. In order to obtain the output relation, a sequence of steps are executed, within each of which the tuples that do not satisfy a part of the predicate are eliminated.

When a constraint is stored, the box consistency of each variable (minimum-bounding hyper-rectangle) in each constraint is also stored in one of the three tables presented in Section 2. This information leads to a more efficient evaluation of the queries.

We address the problem of an efficient query evaluation of the Selection Operator that follows a set of steps which is the result of an exhaustive analysis of the problem. These steps are determined by the type of attributes involved, which include:

1. **Classic attribute:** For an input relation R, the predicate over the classic attributes has to be analysed according to relational algebra, and a new constraint relation must be obtained. This is possible since the constraints and relational data can be differentiated.

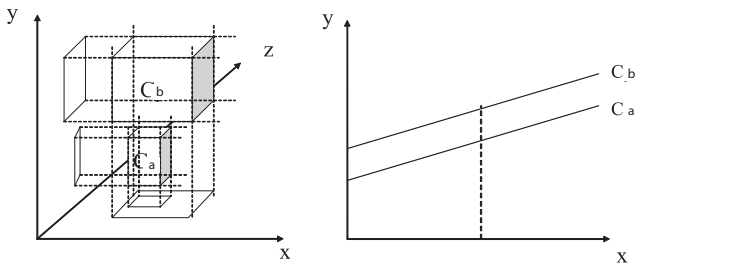

Fig. 5. Example of selection over certain variables.

2. **Constraint-variable attribute:** The second step is related to constraint-variable attributes, where these attributes represent numeric variables whose maximum and minimum values are stored. The implementation of this type of attribute is analysed in Subsection 5.1.
3. **Constraint attribute:** For the constraint attributes, two types of treatments are used, box consistency and the solving of CSPs. The implementation of this type of attribute is analysed in SubSection 5.2.

Classically, the object retrieval for spatio-temporal objects is formed of two steps: The filter step to quickly prune the indexed objects, and the refinement step where the exact representations of candidates using computational geometry algorithms are inspected. In our proposal, the filter step is executed by means of the box consistency analysis. There are a significant number of papers that study the benefits of the use of the box consistency and how to improve the evaluation time based on interval-based solvers (Chabert & Jaulin, 2009; Araya, Trombettoni, & Neveu, 2010; Trombettoni, Papegay, Chabert, & Pourtallier, 2010; Chabert, Trombettoni, & Neveu, 2005). There exist other proposals, such as the use of internal rectangles (Lin & Tan, 2003), but they cannot be used directly in our problem because a constraint can represent more than one object, and hence the internal rectangle indexation would need adapting. Reduction of the space of analysis for two dimensions has been proposed in papers such as Brisaboa, Luaces, Navarro, and Seco (2010), Brisaboa, Luaces, Navarro, and Seco (2013). Our proposal of a refinement step is based on the use of Constraint Programming to support any type of problem or number of variables, thereby obviating the made-to measure algorithm, which depends on the type of problem to solve and the number of variables.

5.1. Evaluation of a constraint-variable attribute in a selection

The predicates related to these attributes can define a comparison between two constraint-variable attributes, a constraint-variable attribute and a classic attribute, or a constraint-variable and a constant attribute. These three types of comparison can be considered as only two types, since the comparison between a constraint-variable and a classic attribute, or a variable and a constant are equivalent. The comparison between a constraint-variable and a classic attribute can be studied as a comparison between a constraint-variable attribute and a constant where the value of the constant is different for each tuple. The various cases are handled by means of the analysis of the box consistency. For example, if the tuples of an output relation have to satisfy a condition where the variable `a` is smaller than the value 5, and the domain of this variable for a constraint is 10..15, then the tuple with this variable does not belong to the output relation. In certain cases, this elimination of tuples can be performed without creating a CSP, simply by carrying out a classic query for classic attributes, since the maximum and minimum value of the variables are stored. However, in order to be sure that the value can be instantiated, a CSP has to be created.

An example of predicate for constraint-variable attributes can be.

$\sigma_{Relation.c>Relation.d}(R)$, where R is the relation shown in Fig. 6. As the output of the selection, the first tuple of the relation is obtained $\{x = 5 \wedge y = 2 \wedge a + b > c \wedge d + a < 8\}$, and since the second and fourth tuples have the variable d, this implies that the relation between variables cannot be defined. With respect to the third tuple, the constraint $d > c * c + 1$ appears, hence the condition $\sigma_{Relation.c>Relation.d}(R)$ cannot be satisfied.

5.2. Evaluation of constraint attributes for data selection

For the constraint attributes, we first propose an analysis of the box consistency in order to determine whether the predicate involved satisfies the condition over the constraint attributes. If this analysis is insufficient, a CSP has to be created and solved.

- **Box consistency analysis:** If the minimum and maximum values that a variable can take are analysed, it is possible to infer whether two constraints satisfy a condition, for example, for the comparison between the constraints C_a and C_b, both defined over the variables v_1 and v_2, where the ranges are $C_a(v_1 : [5..15], v_2 : [20..30])$ and $C_b(v_1 : [20..25], v_2 : [40..55])$. If the predicate is $C_a < C_b$, then it is possible to ensure that the evaluation of the comparison is *true*. However if the predicate is $C_a \subseteq C_b$, then the evaluation of the comparison is *false*. All the combinations are studied in depth in this section.
- **Building CSP:** Although there are cases where it is possible to determine whether a tuple belongs to an output relation using only the analysis of the box consistency, other cases do exist where a CSP has to be built and solved. Returning to the previous example, if the domain of the variables is $C_a(v_1 : [5..15], v_2 : [20..30])$ and $C_b(v_1 : [2..25], v_2 : [10..55])$, and the predicate is $C_a \subseteq C_b$, although the possible solutions of the variables of C_a are included in the possible solution of the variables of C_b, it is not possible to ensure that the predicate is satisfiable. An example where the ranges of sets of variables are included in another constraint, but not the solutions, is shown in Fig. 7. In these cases, it is necessary to create and solve CSPs with both constraints as explained in SubSection 3.2.

In the work of Veltri (2001), the idea of approaching the constraints using box consistency is introduced in order to improve the evaluation time in the query evaluation for CDBs. Despite the limitations of that study, it defines the operations and types of relation between linear constraints to represent polygons with only two dimensions. Our approach is based on this proposal but is defined for all types of polynomial constraints, not only polygons, and for any number of dimensions. Moreover, the aforementioned paper by Veltri adds the description of each minimum-bounding rectangle in each constraint with the $\wedge$ Boolean operator, while our proposal stores the minimum and maximum values of the minimum-bounding rectangle as classic numeric data with the characteristics of relational algebra. This way of storing the box consistency obviates the building and solving of CSPs for the

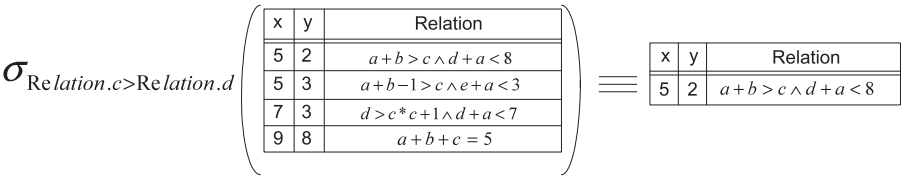

Fig. 6. Example of the Selection Operator with constraint-variable attributes in the predicate.

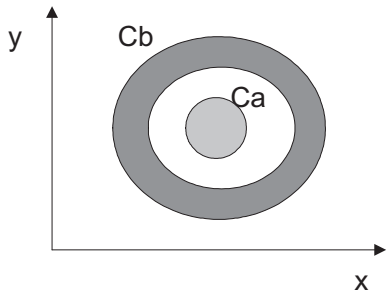

Fig. 7. Example of $C_a \not\subseteq C_b$.

tuples when the box consistency can infer the evaluation of the query. The various types of relations between two constraints related to the ranges of their variables (shown in Fig. 8) include:

- **(a)** All the possible values of a variable v for the constraints C_a are always smaller than the values of the variable v of the constraint C_b.
- **(b)** The greatest value of a variable v for the constraint C_a is equal to the smallest value of this variable in the constraint C_b.
- **(c)** A variable v can take a set of values that satisfy both constraints C_a and C_b, and the values of v satisfying C_a are smaller than the values satisfying C_b.
- **(d)** For the variable v, all the values for the constraint C_a are also solutions of C_b, although C_b can have more solutions than can C_a.
- **(e)** All the possible values of v to satisfy C_a are also values that satisfy C_b, and vice versa.

These relations between domains can help towards determining the relations between constraints as described in Section 3.2, since modelling and solving CSPs is rendered unnecessary in some cases. With a generalization for n variables, $\text{Min\_Value}_{C_j}(v_i)$ represents the minimum value of the variable v_i that satisfies constraint C_j, which can be $-\infty$ or a numeric value, and $\text{Max\_Value}_{C_j}(v_i)$ represents the maximum value of the variable v_i that satisfies constraint C_j, which can be $+\infty$ or a numeric value. Let $v_1 \ldots v_n$ be the variables of the constraints C_a and C_b, and let $\text{Min\_Value}_{C_a}(v_1), \ldots,$ $\text{Min\_Value}_{C_a}(v_n)$ be the minimum values of the variables of the constraint C_a. Let $\text{Max\_Value}_{C_a}(v_1), \ldots,$ $\text{Max\_Value}_{C_a}(v_n)$ be the maximum values of each variable for the constraint C_a, let $\text{Min\_Value}_{C_a}(v_1), \ldots,$ $\text{Min\_Value}_{C_b}(v_n)$ be the minimum values of the variables of the constraint C_b, and let $\text{Max\_Value}_{C_b}(v_1), \ldots,$ $\text{Max\_Value}_{C_b}(v_n)$ be the maximum values of the variables for the constraint C_b. In the cases where the clause `FOR` is included in the condition, in order to determine the subset of variables involved in the comparison, only these variables are analysed. If the key words `AND` and `FOR` are used, the semantic equality between the variables is established. The various cases depend on the operation between the constraints C_a and C_b, and the ranges of their variables are shown in Table 1. This table represents the evaluation of the comparison between two constraints (C_a and C_b), depending on the relation of their box consistencies. Sometimes it is possible to know if the comparison will be *true* or *false*, although sometimes will be necessary to build a CSP, as explained in SubSection 3.2.

Table 1
Evaluation of comparison using box consistency prototypes.

C_a, C_b	Fig. 8(a)	Fig. 8(b)	Fig. 8(c)	Fig. 8(d)	Fig. 8(e)
$\_<\_$	true	false	CSP	CSP	false
$\_\leqslant\_$	true	true	CSP	CSP	false
$\_\{>, \geqslant\}\_$	false	false	false	CSP	false
$\_=\_$	false	false	false	false	CSP
$\_<>\_$	true	true	true	true	CSP
$\_\&\_$	false	CSP	CSP	CSP	CSP
$\_\{\subset, \subseteq, \supset, \supseteq\}\_$	false	false	false	CSP	CSP

In Fig. 9, some examples of relations of satisfaction using box consistency are shown, highlighting two constraints with a shaded area. In Fig. 9a, an example is given where $C_a < C_b$ is satisfiable, which means that all the values of the variables v_1 and v_2 that satisfy C_a, are smaller than the values of these same variables that satisfy C_b. In Fig. 9b, an example is presented where $C_a < C_b$ FOR $\{v_2\}$ is satisfiable, since all the values of v_2 that satisfy C_a are smaller than the values of this variable that satisfy C_b.

Fig. 10a presents an example where the projection obtained from the box consistency of C_a is included in the projection of the box consistency of C_b, and where all the solutions of C_a are also solutions of C_b. On the other hand, in Fig. 10b, an example is presented where, although the range of the variables for C_a is included in the range of C_b, not all the solutions of C_a are solutions of C_b. The example shown in Fig. 10c is similar, since the range of the variables for C_a is equal to the range of C_b, but not all the solutions of C_b are solutions of C_a. For the *inclusion* and *equal* operations, by means of using the ranges it is possible to ensure that two constraints do not satisfy a condition. However, to ensure that the constraints satisfy the condition, it is necessary to build and solve a CSP.

If there is no *FOR* clause, then this analysis is performed for all the variables, either until one of them fails to satisfy a condition, or until all the variables had been analysed. If finally the study returns *true*, then the tuples, where the constraints are, belong to the output relation.

5.3. Scope of applicability of our proposal

As a consequence of the model presented and the strategies of evaluation, the scope of application of our proposal is limited by three characteristics:

- **Constraint Programming.** We have decided to use Constraint Programming since: it is a very mature area that has been applied to a wide range of problems, and with high level of complexity; it uses propagation techniques to reduce the search space in an efficient way; there are numerous tools and algorithms to model and solve problems, and; it permits an easy

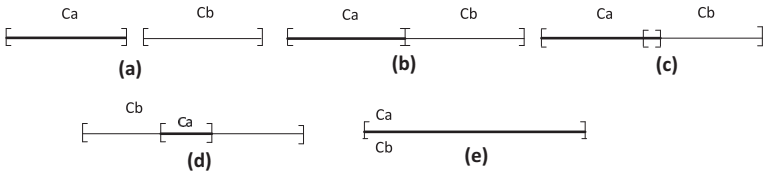

Fig. 8. Types of relations between the ranges of the variables of two constraints.

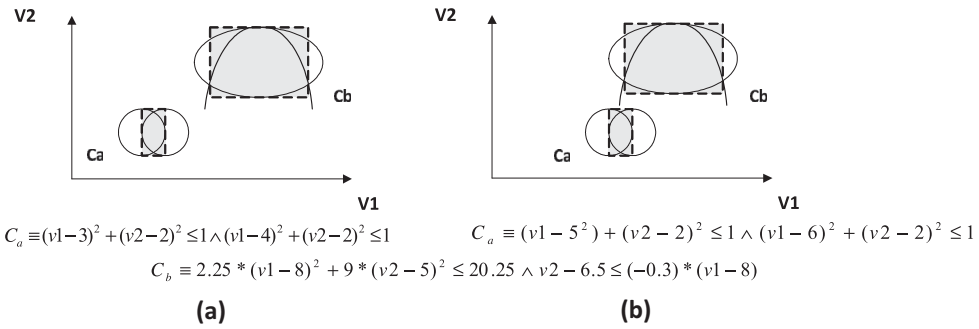

Fig. 9. Example of relations between constraints by means of their box consistency.

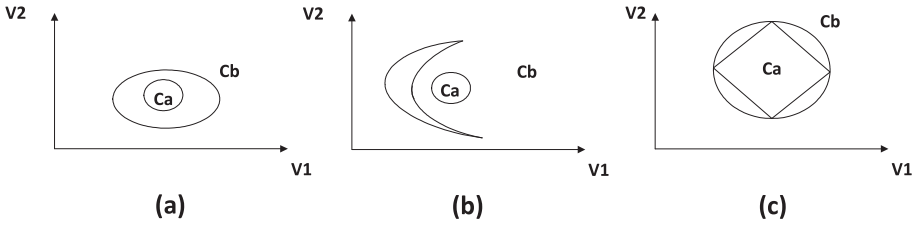

Fig. 10. Example of constraints with included ranges.

definition of the complex data using a wide range of constraints, such as implication constraints, disjunctive constraints, reified constraints, global constraints, and channelling constraints.

- **The range of Constraints.** This limitation is centred on the capacity of their expressiveness as allowed by the grammar and type of variables. The grammar of the Constraints constitutes the formal representation of the relations between the data involved. The limitations of use of the proposal appear when the information to be stored cannot be represented by numerical constraints over the defined domains, or the operator is not included in this proposal, such as when a relation among two variables is described by means of a trigonometric function. The limitation of the data domain and the operations that can be employed are established by the solver used for the Constraint Programming Problem, explained in Section 4. Most of the commercial solvers maintain the capacity to include Float, Integer, Sets, and Boolean variables in the model, thereby making it possible to cover a significant number of problems and their business data constraints.
- **Complexity of resolution of the CSP.** In the worst case scenario, the evaluation time depends on the complexity of the resolution of the CSP. This has been analysed in great depth over recent decades (Cheeseman, Kanefsky, & Taylor, 1991), and depends on two parameters: the width of the graph and the order parameter. On one hand, the width of the graph represents the relation between the constraints: the tractability in CSPs is due to the structure of the constraint network, where the tree-structured CSPs have polynomial complexity (linear with respect to the number of variables, and quadratic with respect to the cardinal of the domain of the variables). On the other hand, the order parameter, defined as the ratio of the number of forbidden tuples to the total number of possible combinations, determines the partition of the problem space into under-constrained, over-constrained and just-constrained problems. In the first two cases, the problems are scalable, but in just-constrained problems, a significant increase in solving cost could occur and the scalability would not be possible (Krzysztof, 2003).

For these reasons, no affirmation about the efficiency or scalability in a generic way can be given by our proposal, since our framework permits any type and number of Constraints defined with numerical variables, and therefore the evaluation time depends on the specific problem. However, in order to reduce the complexity and allow the scalability of the problems, we have proposed a set of strategies shown experimentally in the following section.

6. Example of use and experimental results for the Selection Operator

In order to extend SQL and include the explained strategies in the expert system, this proposal has been implemented as a Java Library component, called CDB_Package, which shields the user from unnecessary details on how the information is stored and how the queries are evaluated. With a very similar code to that used for any database connection in Java™, the user defines the relational database connection, writes the sentence and executes the query. The difference is that this package includes a CDB class that creates, if they have not been created before, the three tables to index the constraints and variables, and establishes the connection with the database and the type of database management system. In the method *executeQuery* of class CDB, the steps described in Section 5 are implemented. The current database management systems permitted are Oracle and MySQL. CDB_Package includes the solver for the CSP used is Rochart and Jussien (xxxx), which is also a Java library for CSPs, although it can be enlarged to other solvers such as JSolver™ (ILOG, xxxx). CDB_Package has been developed and used in the Victor R&D Project (Victor-Project, 2013).

```
Connection con = DriverManager.getConnector
("jdbc...", ...)
String cmd = "Select ....";
CDB bd = new CDB (con, "Oracle");
PreparedStatement stmt = db.executeQuery (cmd);
```

Based on the Victor Project, we have tested how the various strategies that we propose reduce the evaluation time. In order to test queries with this Selection Operator, two different tables are used to represent the various zones of construction and the static objects modelled. The Victor Project only contains two-dimensional objects, but the same methodology can be used for any

number of variables or problems that can be represented by constraints. Table *Objects* consists of 550 tuples, while Table Zones has 100 tuples. The two tables have two classic attributes to represent their identification (*id*) and description (*desc*), and a constraint attribute (location) to represent the location with respect to the constraint-variables *x* and *y* for the example.

All the measures presented in this paper relate our solution with the improvements proposed herein, and not with other solutions since there are no other proposals that permit polynomial constraints to be managed, as is explained in Section 7. All the measures are presented in milliseconds and have been obtained by means of an AMD Athlon^{TM} 64 Dual Core Processor 4200 + 2.21 GHz, where the Database Management System Oracle 10 g is running. Various query evaluation times are presented in Fig. 11. It is also shown how, due to the relational structure of the information, the evaluation time decreases when relational algebra is used for classic data, and where the creation of CSPs is rendered unnecessary thanks to the use of box-consistency analysis.

We have taken one hundred measurements for each type of Selection Operator, combined by using comparison of variables (FOR C_1.x = C_2.x') as explained in Section 4. Figs. 11 and 12 represent, for each type of test, the minimum, maximum, and average evaluation time for the tests, and the number of CSPs created respectively. For example, case B presents the evaluation of a query to evaluate which static objects are to the North of a given object. If all the comparisons between the objects are analysed, then the potential combinations of objects is $302,500$ ($550*550$) independent of which object is given. Therefore $302,500$ CSPs must be created (as presented in Fig. 12), which expends the evaluation time to 1.8 10^8 milliseconds as shown in Fig. 11. If relational algebra is used and only one object is compared with the rest, only 550 CSPs are created (Fig. 12), thereby decreasing the evaluation time (Fig. 11) to an average of 1.8 10^6 milliseconds for the example. In the case when box consistency is used, the number of CSPs can be even zero, and the evaluation time depends on the constraints involved in the comparison, to an average of 4.64 10^5 milliseconds for the example.

Since the comparison operators share certain similarities, they can be grouped into the following four sets $\{<, \leqslant, >, \geqslant\}$, $\{\subset, \subseteq, \supset, \supseteq\}$, {&}, $\{<>, =\}$. We present tests for only the first three sets, since the last set represents simpler operators that fail to contribute further information to the conclusions. For each set, we propose the three variants described depending on the variables used (for all the variables, for a set of variables, or for a set of variables with an equality relation).

(A) The comparison operators $<, \leqslant, >$ and $\geqslant$ are found to be equivalent in complexity, therefore the tests were run for the comparison operator $<$ comparing different objects, in order to determine which objects lie to the northeast of the object with the identification *k* (have higher values for *y* and *x* variables). The query is: `Select o2.id from Objects o1, Objects o2 where o1.location` $\leqslant$ `o2.location AND o1.id = k`.

(B) The comparison operators $<, \leqslant, >$ and $\geqslant$ can be combined with the "FOR" operator to determine which variables are compared. The tests were run for the comparison operator $<$ for different objects, in order to discover which objects lie to the North of the object with the identification *k* (have higher values for *y* variables). The query is: `Select o2.id from Objects o1, Objects o2 where o1.id = k and o1.location` $\leqslant$ `o2.location FOR {y}`.

(C) The comparison operators $<, \leqslant, >$ and $\geqslant$ can be combined with the "FOR" operator to determine which variables are compared and where another variable is semantically equal in the constraints involved. The tests were run for the comparison operator $<$ for different objects, in order to discover which objects have values (for *y* and x variables where both objects are equal) higher than those of the object with the identification *k*. The query is: `Select o2.id from Objects o1, Objects o2 where o1.id = k and o1.location` $\leqslant$ `o2.location FOR {y and o1.x = o2.x}`.

(D) The comparison operators $\subset, \subseteq, \supset$ and $\supseteq$ are equivalent in complexity, and hence the tests were run for $\subset$ for different objects in order to determine which objects are included (for the variables *y* and *x*) in the zone with identification *z*. The query is: `Select o.id from Objects o, Zones z1 where z1.id = z and o.location` $\subset$ `z1.location`.

(E) The comparison operators $\subset, \subseteq, \supset$ and $\supseteq$ are equivalent in complexity, and hence the tests were ran for $\subset$ for different objects in order to find out which objects are included (for the variables *y* and *x*) in the zone with identification *z*. The query is: `Select o1.id from Objects o1, Zones z1 where z1.id = z and o1.location` $\subset$ `z1.location {FOR y}`.

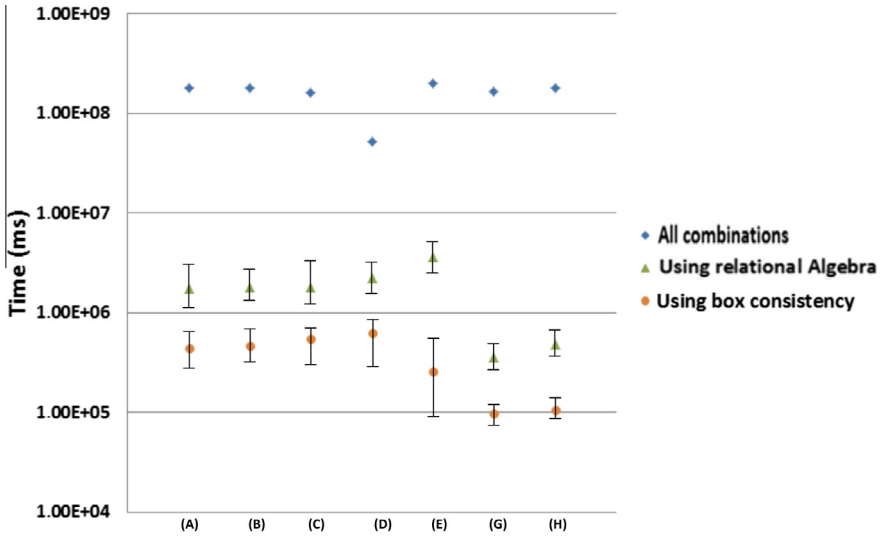

Fig. 11. Evaluation time for query examples.

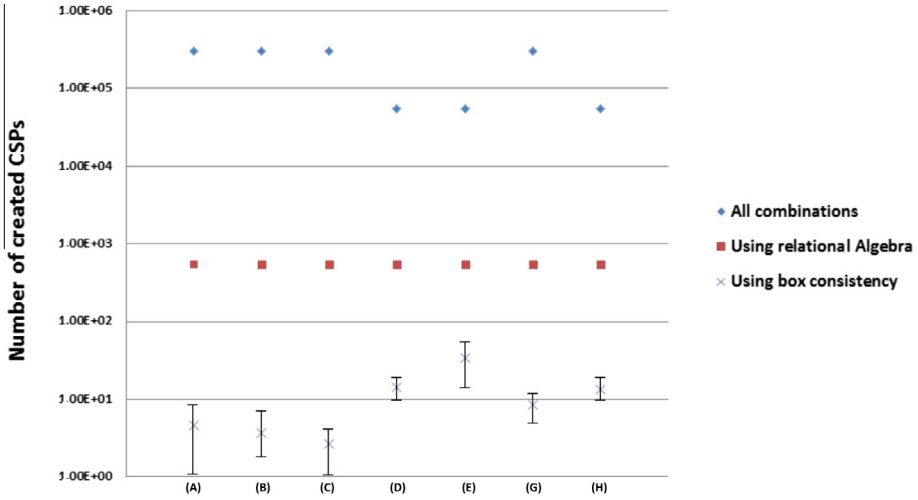

Fig. 12. Number of CSPs created for query examples.

(F) The comparison operators $\subset$, $\subseteq$, $\supset$ and $\supseteq$ need no evaluation for the operator `FOR y and x = x'` since it makes no sense. If the sentence $C_a \subseteq C_b$ FOR y is satisfiable, then $C_a \subseteq C_b$ `FOR y and` C_a`.x = `C_b`.x` is also satisfiable.

(G) The comparison operator & describes if there is at least one value in common between two constraints. For example, it can be used to obtain if there is any object that touches to another object. The query is: `Select o.id from Objects o1, Object o2 where o1.location & o2.location`.

(H) The comparison operator & can also be used for some variables. It can, for example, be used to obtain the object whose projection for the variable x has some values in common with a zone with the identification *z*. The query is: `Select o.id from Objects o, Zones z1 where z1.id = z and o.location & z1.location FOR {`y`}`.

(I) The comparison operator & needs no evaluation for the operator `FOR y and x = x'` since it makes no sense. If the sentence C_a & C_b `FOR y` is satisfiable, then C_a & C_b `FOR y and` C_a`.x = `C_b`.x` is also satisfiable.

As conclusions of Figs. 11 and 12, it should be pointed out that the proposed solution improves the evaluation time since relational algebra allows us to select the related tuples depending on the query, and the box consistency analysis renders the creation and solution of the CSPs unnecessary. Although there is a major reduction in the number of CSPs, the query evaluation time does not decrease by the same percentage. It is produced by the time expended for the database access, since although several CSPs are created they are very easy to solve, and they can be evaluated in a very short time.

7. Related work

The problem of the management of complex data implies two areas of analysis: how to store and represent the information, and how to filter the objects to improve the evaluation time.

The main proposals found in the literature are based on geometric data, and therefore need to represent two and three dimensions, including: Segments; triangles (Tøssebro & Nygård, 2011;Tøssebro & Nygård, 2006); and Realms, which are a set of points and non-intersecting line segments over a discrete domain, that is, a grid (Gting & Schneider, 1993). The geometric data selection can be improved applying grid partitioning to reduce the query evaluation time (Park, 2014). All these proposals are based on spatial databases storing geometric data, but they cannot be used for several variables or for general purpose databases. In order to overcome this limitation, we have decided to use Constraints to represent the relation of the variables, and we have been inspired by their algorithms to improve the evaluation time (Lin & Tan, 2003) used to index spatial objects. To this end, constraint data has been proposed as a more general purpose application.

There are several proposals for implementing and building prototypes that store and select constraint data, most of which are based on the scenario of CDBs. The most significant examples of these are analysed below:

- **DISCO** (Revesz, 2000) (DATALOG with Integer Set COnstraints) is a CDB system which implements Datalog with Boolean constraints on variables that range over finite and infinite sets of integers. The constraints in DISCO are stored in a file using facts and rules, thereby ignoring the advantages that relational databases offer, despite the fact that a large amount of research on relational databases has been published over the years, such as in the design of large-scale databases and concurrent access. The syntax of DISCO is very similar to that of Datalog, and very different from the standard SQL. DISCO is presented as an extensible system where the Boolean algebra can be modified by replacing certain operators in a straightforward way.
- **MLPQ/PReSTO** (Revesz, 2008;Revesz, 2010) presents a combination of MLPQ (Management of Linear Programming Queries) and PReSTO (Parametric Rectangle Spatio Temporal Object). MLPQ is a system to manage and query linear constraints in a CDB. It allows Datalog queries and the addition of operators over linear functions. PReSTO enables query systems which change over time, with similar semantics to that of classic relational algebra. Although both present a similar SQL syntax, they actually use plain text to store the information and a query transformation process into Datalog.
- **DEDALE** (Grumbach, Rigaux, Scholl, & Segoufin, 2000) is an implementation of CDBs based on linear constraint models. DEDALE proposes a language to query CDBs, which permits information to be obtained by using a graphical interface to show the results. In order to represent the constraints, DEDALE uses the object-oriented paradigm, which is an appropriate way to represent complex data. The disadvantage in this proposal is that all the information is stored as objects, and therefore fails to take advantage of relational algebra.

The syntax of DEDALE is very different from standard SQL, and therefore any user of the system must learn another language, which is a great disadvantage.

- **CCUBE** (Brodsky, Segal, Chen, & Exarkhopoulo, 1999) (Constraint Comprehension Calculus) is the first constraint object-oriented database system. The CCUBE system was designed to be used for the implementation and optimization of high-level constraint object-oriented query languages. The CCUBE data manipulation language, Constraint Comprehension Calculus, is an integration of constraint calculus for extensible constraint domains within monoid comprehension. CCUBE serves as an optimization-level language for object-oriented queries. The data model for the constraint calculus is based on Constraint SpatioTemporal (CST) objects. CCUBE guarantees polynomial time data complexity whose implementation uses the linear programming package CPLEX developed by Bixby et al. (1998). CPLEX allows both integer programming and very large linear programming problems to be solved.
- **CQA/CDB** is a CDB prototype whose constraints are linear over rational coefficients (Goldin, Kutlu, & Song, 2003;Goldin, 2004). This solution uses only linear constraints in order to represent and process the information, since it studies the spatiotemporal area, whose data can be approximated with linear constraints. The constraint tuples are represented by a matrix of coefficients. In this prototype, queries are based on relational operators, and look similar to SQL; however there are numerous syntactic and semantic differences to the standard. CQA/CDB does not allow new constraints to be inferred by using the constraints stored in the CDB.
- **LORCDB** (Labelled Object-Relational Constraint Database) (Gómez-López, Gasca, Valle, & De la Rosa, 2005;Gómez-López et al., 2009) presents an extension of an object relational database based on the use of Constraint Programming paradigm. This proposal has certain extension of the SQl syntax, but only for the projection operator. As the constraints are represented by means of objects, both linear and polynomial constraints can be described.

The various characteristics are shown in Table 2. The reason why we have decided to extend the LORCDB is because the rest of the proposals mentioned above have the following disadvantages:

- Some of the previous prototypes offer languages that are centred on a specific application domain. Most of these prototypes are developed to work with spatiotemporal data such as those in Deo (2002), Toman (2000). In this paper we have presented a generic example, where the language is not dependent on the semantics of the problem and constraint data modelling can easily be carried out since we have the possibilities of a classic relational database, adding the management of constraints.
- There are many query languages whose syntax greatly differ from that of SQL, thereby causing the user to need to learn a new language. Sometimes this syntax is focused on the application of CDBs, frequently in the spatiotemporal domain.
- Proposals, such as MLPQ, do not use relational databases, thereby losing out on the relational theory advantages, such as uniqueness of the primary key, and referential integrity. Some of these proposals are based on Deductive Databases whose disadvantages of inconsistency is analysed in Mayol and Teniente (2003). A vast amount of experience and theories in relational databases have improved the selection of data by using indexation, clustering, and hashing, all of which have been included in commercial relational database management systems, except for the case when Deductive Databases are used.
- These proposals only handle linear constraints, since, for spatio-temporal problems, this type of constraint is good enough to approximate the problem, and solving linear constraint problems is more efficient than solving problems for polynomial constraints. This approximation is inappropriate for applications where more precision is necessary.
- None of these prototypes defines *Constraints* as a new type of data, and although some use the object-oriented paradigm, they fail to distinguish between discrete and continuous information, and hence are unable to exploit relational algebra in any of these aspects. Our solution is independent of the database management system used, it implies that it is also independent of the format. There is no a file format where the constraints are stored. The use of CDB_Package Java library is more versatile than the other options since it is adaptable to any commercial database (MySQL, Oracle, PostGreSQL, ...). This implies that any database can be "transformed" into a Constraint Database simply by using the CDB_Package.
- A paper where the combination of constraints and relational databases is proposed (Cai, 2004), but presents many limitations solved in LORCDB. Examples of these limitations include: all the constraints stored in the same column have to be defined over the same set of variables, only linear constraints can be stored and queried; there is nether indexation nor box-consistency analysis to improve the evaluation time; and the condition of the Selection Operator is not adapted to constraint-attributes. Other proposals, such as Talebi, Chirkova, and Fathi (2013), use Constraint Programming to improve the selection of data, but for extracting classic data not for complex data.

8. Conclusions and future work

In this paper, the SQL language has been extended in order to improve the data selection for complex data represented, by means of constraints for any number of variables and different types of data domains. Our proposal enables a set of strategies based on Constraint Programming, to be included into a relational database. It permits an extended relational database to work as an expert system for queries of a more complex nature, while shielding the user from unnecessary details about how the query is evaluated. The Selection Operator enables us to determine which constraints satisfy a condition with respect to other constraints, where the meaning of the comparison depends on the case where it is used,

Table 2
Characteristics of the CDB prototypes.

	Logic model	Logic progr.	SQL syntax	Constraint type	Easy constraint data modelling	Independent format
MLPQ/PReSTO	File	Datalog	Yes	Linear	High	No
DISCO	File	Datalog	No	Linear	Medium	No
DEDALE	BDOO	No	Yes	Linear	Medium	No
CCUBE	BDOO	No	Yes	Linear	Medium	No
CQA/CDB	?	No	Yes	Linear	Medium	No
LORCDB	BDOR	No	Yes	Linear and polynomial	High	Yes

and not exclusively on topological relations. The proposal is based on the storage of relational and constraint data in a relational database, by defining a set of strategies based on relational algebra, CSPs and box consistency to minimize the evaluation time, since the Constraint Programming paradigm is more efficient and general. Furthermore, this proposal permits constraints to be treated as classic data in the general purpose databases, thereby helping the user to select data from an extended relational database.

The paper contributes towards the existing literature in the following ways: a general purpose language based on SQL is proposed; linear and polynomial constraints are supported for storage and evaluation; classic and complex data can be combined in the same database and in the same queries; and a set of strategies has been proposed for the reduction of the evaluation time. Moreover, our proposal is adaptable to any commercial database, since it is independent of the database management system. The use of CDB_Package Java library is more versatile than all other options since it is adaptable to any commercial database (MySQL, Oracle, PostGreSQL, etc). The applicability of our proposal is demonstrated by means of its use in a real project.

The limitations of use of the proposal appear when the constraints cannot be represented by numerical relations, data type, or by the operators included in the proposed grammar, such as when a relation between two variables is described by means of a trigonometric function. The limitation of the data domain and the operations that can be employed are established by the solver used for the Constraint Programming Problem, explained in Section 4. However, commercial solvers tend to provide more complex constraints, such as *If. . .then*, *For all*, and *All different*.

As future work, we propose redefining other operations of SQL, such as cartesian product, union, difference and join. We also propose the adaptation of other index structures, such as R-trees and internal rectangles for constraints as an improvement to the current proposal.

Acknowledgment

This work has been partially funded by the Junta de Andalucía by means of la Consejería de Innovación, Ciencia y Empresa (P08-TIC-04095) and by the Ministry of Science and Technology of Spain (TIN2009–13714) and the European Regional Development Fund (ERDF/FEDER).

References

Araya, I., Trombettoni, G., & Neveu, B. (2010). Making adaptive an interval constraint propagation algorithm exploiting monotonicity. In *Proceedings of the 16th international conference on Principles and practice of constraint programming, CP'10* (pp. 61–68). Berlin, Heidelberg: Springer-Verlag.

Bixby, R. E., Boyd, E. A., & Ríos-Mercado, R. Z. (Eds.). (1998). *Integer Programming and Combinatorial Optimization, 6th International IPCO Conference, Houston, Texas, USA, June 22-24, 1998, Proceedings. Lecture Notes in Computer Science* (Vol. 42). Springer.

Brisaboa, N. R., Luaces, M. R., Navarro, G., & Seco, D. (2010). Range queries over a compact representation of minimum bounding rectangles. In *Proceedings of the 2010 international conference on Advances in conceptual modeling: applications and challenges, ER'10* (pp. 33–42). Berlin, Heidelberg: Springer-Verlag.

Brisaboa, N. R., Luaces, M. R., Navarro, G., & Seco, D. (2013). Space-efficient representations of rectangle datasets supporting orthogonal range querying. *Information Systems, 38*(5), 635–655.

Brodsky, A., Segal, V. E., Chen, J., & Exarkhopoulo, P. A. (1999). The CCUBE Constraint Object-Oriented Database System. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data* (pp. 577–579). ACM Press.

Cai, M., 2004. Integrating constraint and relational database systems, in: CDB, pp. 180–188.

Chabert, G., Trombettoni, G., Neveu, B., 2005. Box-set consistency for interval-based constraint problems, in: SAC, pp. 1439–1443.

Chabert, G., & Jaulin, L. (2009). Hull consistency under monotonicity. In *Proceedings of the 15th international conference on Principles and practice of constraint programming, CP'09* (pp. 188–195). Berlin, Heidelberg: Springer-Verlag.

Cheeseman, P., Kanefsky, B., & Taylor, W. M. (1991). Where the really hard problems are. *Proceedings of the 12th international joint conference on Artificial intelligence - IJCAI'91* (Vol. 1, pp. 331–337). San Francisco, CA, USA: Morgan Kaufman Publishers Inc..

Dechter, R. (2003). *Constraint Processing (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufman.

Deo, A.D., 2002. Modeling Spatial and Temporal in an Object-Oriented Constraint Databases, (Ph.D. thesis). von der Fakultt IV-Elektrotechnik und Informatik der Technish Universitt zur Erlangung des akademischem Grades, Berln.

Goldin, D. (2004). Taking Constraints out of Constraint Databases.. In *Constraint DataBases. Lecture Notes in Computer Science* (Vol. 74, pp. 168–179). Springer.

Goldin, D., Kutlu, A., & Song, M. (2003). Extending the constraint database framework. In *PCK50* (pp. 42–54). New York, USA: ACM Press.

Gómez-López, M. T., Ceballos, R., Gasca, R. M., & Valle, C. D. (2009). Developing a labelled object-relational constraint database architecture for the projection operator. *Data & Knowledge Engineering, 68*(1), 146–172.

Gómez-López, M.T., Gasca, R.M., Reina-Quintero, A., 2011. Model-driven engineering for constraint database query evaluation, in: Workshop Model-Driven Engineering, Logic and Optimization: friends or foes?, MELO 2011.

Gómez-López, M. T., Gasca, R. M., Valle, C. D., & De la Rosa, F. (2005). Querying a polynomial constraint object-relational database in model-based diagnosis. *Lecture Notes in Computer Science, 3588*, 848–857.

Granvilliers, L., Goualard, F., & Benhamou, F. (1999). Box consistency through weak box consistency. In *ICTAI '99: Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence* (pp. 373). Washington, DC, USA: IEEE Computer Society.

Grumbach, S., Rigaux, P., Scholl, M., & Segoufin, L. (2000). The DEDALE prototype. *Constraint Databases*, 365–382.

Gting, R. H., & Schneider, M. (1993). Realms: A foundation for spatial data types in database systems. In *3RD INT. SYMP. ON ADVANCES IN SPATIAL DATABASES, LNCS 692* (pp. 14–35). Springer-Verlag.

ILOG, xxxx. Jsolver 2.1, Reference Manual April.

Kanellakis, G. M. K. P. C., & Revesz, P. Z. (1990). Constraint query languages. *Symposium on Principles of Database Systems*, 299–313.

Krzysztof, A. (2003). *Principles of Constraint Programming*. New York, NY, USA: Cambridge University Press. Ed..

Lee, K. C. K., Unger, B., Zheng, B., & Lee, W.-C. (2011). Location-dependent spatial query containment. *Data & Knowledge Engineering, 70*(10), 842–865.

Lin, P. L., & Tan, W. H. (2003). An efficient method for the retrieval of objects by topological relations in spatial database systems. *Information Processing & Management, 39*(4), 543–559.

Malpica, J. A., Alonso, M. C., & Sanz, M. A. (2007). Dempster-shafer theory in geographic information systems: A survey. *Expert Systems with Applications, 32*(1), 47–55.

Marriott, K., & Stuckey, P. J. (1998). *Programming with Constraints. An introduction, Simplification, Optimization and Implication*. The Mit Press.

Mayol, E., & Teniente, E. (2003). *Consistency preserving updates in Deductive Databases* (Vol. 47). Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B.V..

Park, K. (2014). Location-based grid-index for spatial query processing. *Expert Systems with Applications, 41*(4), 1294–1300.

Poelmans, J., Ignatov, D. I., Kuznetsov, S. O., & Dedene, G. (2013a). Formal concept analysis in knowledge processing: A survey on applications. *Expert Systems with Applications, 40*(16), 6538–6560.

Poelmans, J., Kuznetsov, S. O., Ignatov, D. I., & Dedene, G. (2013b). Formal Concept Analysis in knowledge processing: A survey on models and techniques. *Expert Systems with Applications, 40*(16), 6601–6623.

Revesz, P.Z., 1995. Datalog Queries of Set Constraint Databases., in: ICDT, pp. 425–438.

Revesz, P. (1998). Safe query languages for constraint databases. *ACM Transactions on Database Systems, 23*(1), 58–99.

Revesz, P. Z. (2000). The DISCO system. *Constraint Databases*, 383–389.

Revesz, P. (2001). *Introduction to Constraint Databases*. Springer.

Revesz, P. Z. (2008). MLPQ spatial constraint database system. *Encyclopedia of GIS*, 661–662.

Revesz, P. Z. (2010). *Introduction to Databases: From Biological to Spatio-Temporal* (1st Edition.). Springer Publishing Company, Incorporated.

Rochart, X.L.G., Jussien, N.. xxxx. Choco. a java constraint programming library, Reference Manual. <http://www.emn.fr/z-info/choco-solver/>.

Talebi, Z. A., Chirkova, R., & Fathi, Y. (2013). An integer programming approach for the view and index selection problem. *Data & Knowledge Engineering, 83*, 111–125.

Toman, D., 2000. SQL/TP: A Temporal Extension of SQL., in: Constraint Databases, Springer, pp. 391–399.

Tøssebro, E., Nygård, M., 2006. Representing topological relationships for moving objects, in: GIScience, pp. 383–399.

Tøssebro, E., & Nygård, M. (2011). Representing topological relationships for spatiotemporal objects. *GeoInformatica, 15*(4), 633–661.

Trombettoni, G., Papegay, Y., Chabert, G., Pourtallier, O., 2010. A box-consistency contractor based on extremal functions, in: CP, pp. 491–498.

Veltri, P. (2001). Constraint database query evaluation with approximation. In *ITCC '01: Proceedings of the International Conference on Information Technology: Coding and Computing* (pp. 634–638). Washington, DC, USA: IEEE Computer Society.

Victor-Project, 2013. <http://estigia.lsi.us.es/victor/>.