A Polynomial Algorithm for Continuous Non-binary Disjunctive CSPs

Miguel A. Salido, Federico Barber
Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia,
46020 Valencia, Spain
{msalido, fbarber}@dsic.upv.es

Abstract
Nowadays, many real problems can be modelled as Constraint Satisfaction Problems (CSPs). Some CSPs are considered non-binary disjunctive CSPs. Many researchers study the problems of deciding consistency for *Disjunctive Linear Relations (DLRs).* In this paper, we propose a new class of constraints called *Extended DLRs* consisting of disjunctions of linear inequalities, linear disequations and non-linear disequations. This new class of constraints extends the class of DLRs. We propose a heuristic algorithm called DPOLYSA that solves Extended DLRs, as a non-binary disjunctive CSP solver. This proposal works on a polyhedron whose vertices are also polyhedra that represent the non-disjunctive problems. We also present a statistical preprocessing step which translates the disjunctive problem into a non-disjunctive and ordered one in each step.

1 Introduction

Nowadays, many researchers have studied linear constraints in operational Research (OR), constraint logic programming (CLP) and constraint databases (CBD). Subclasses of linear constraints over the reals have also been studied in temporal reasoning, [2, 7], where the main objectives are to study the consistency of a set of binary temporal constraints, to perform value elimination and to compute the minimal constraints between each pair of variables.

Many real problems can be represented as disjunctive linear relations (DLRs) over the reals [2, 7]. The problem of deciding consistency for an arbitrary set of DLRs is NP-complete [12]. It is very interesting to discover classes of DLRs for which consistency can be decided in PTIME [7].

In [8], Lassez and McAloon studied the class of *generalized linear constraints*, this includes linear inequalities (e.g., $x_1+2x_3-x_4 \leq 4$) and disjunctions of linear disequations (e.g., $3x_1 - 4x_2 - 2x_3 \neq 4 \vee x_1 + 3x_2 - x_4 \neq 6$). They proved that the problem consistency for this class can be solved in polynomial time.

Koubarakis in [7] extends the class of *generalizad linear constraints* to include disjunctions with an unlimited number of disequations and *at most one* inequality per disjunction. (e.g., $3x_1 - 4x_2 - 2x_3 \leq 4 \vee x_1 + 3x_2 - x_4 \neq 6 \vee x_1 + x_3 + x_4 \neq 9$). This class is called *Horn constraints.* He proved that deciding consistency for this class can be done in polynomial time.

In [4], Jonsson and Bäckströn present a new formalism called *Horn Disjunctive Linear Relations (Horn DLRs)* that extends the class of Horn constraints since instead of managing only one inequality per disjunction, Horn DLRs can manage one linear relation of the form $\alpha r \beta$ where α, β are linear polynomials and $r \in \{<, \leq, =, \geq, >\}$ per disjunction (e.g., $3x_1 - 2x_3 = 4 \vee x_1 + x_2 - 2x_4 \neq 6 \vee 2x_1 - x_2 + x_3 \neq 5$).

In this paper, we extend DLRs to include disjunctions with an arbitrary number of linear inequalities, linear disequations and non-linear disequations. For example:

$$2x_1 + 3x_2 - x_3 \leq 6 \;\vee\; x_1 + 5x_4 \leq 9 \;\vee\; 2x_1 - x_3 + x_5 \neq 3 \;\vee\; x_2^3 - \sqrt[3]{x_3} \neq 2$$

The resulting class will be called the class of *Extended DLRs*. Moreover, our objective is not only to decide consistency, but also to obtain one or several solutions and to obtain the minimal domain of the variables.

It is well known that these objectives only can be achieved in exponential time. In an attempt to achieve these objectives in polynomial time, we propose an incomplete algorithm called "Disjunctive Polyhedron Search Algorithm" (DPOLYSA) that manages Extended DLRs as a disjunctive non-binary CSP solver. DPOLYSA is a polynomial heuristic algorithm that solves Extended DLRs in most of the times. This algorithm runs a preprocessing step in which the algorithm translates the disjunctive problem into a set of non-disjunctive ones generated by means of a technique based on sampling from finite populations. The set of constraints of each non-disjunctive problem is ordered in ascending order with respect to the number of vertices satisfied in a hypothetical polyhedron. Furthermore, when the problem is reduced to a non-disjunctive one, DPOLYSA applies a non-disjunctive CSP solver called POLYSA [11]. POLYSA manages the non-disjunctive CSP creating a polyhedron by means of the Cartesian Product of some variable domain bounds. This Cartesian Product generates a polyhedron with n^2 random vertices in each polyhedron face. Therefore, the computational complexity is $O(n^3)$. DPOLYSA efficiently manages non-binary CSPs with many variables, many disjunctive constraints and very large domains. This proposal overcomes some of the weaknesses of other typical techniques, like *Disjunctive Forward-Checking* and *Disjunctive Real Full Look-ahead*, since its complexity does not change when the domain size and the number of atomic constraints increase.

2 Preliminaries

Briefly, a disjunctive constraint satisfaction problem (DCSP) that DPOLYSA manages consists of:

- A set of variables $X = \{x_1, ..., x_n\}$.
- A continuous domain of values D_i for each variable $x_i \in X$.
- A set of disjunctive constraints $C = \{c_1, ..., c_p\}$ restricting the values that the variables can simultaneously take.

A solution to a DCSP is an assignment of a value from its domain to every variable, so that at least one constraint per disjunction is satisfied. The objective in a DCSP may be: to determine whether a solution exists; to find one solution, many or all solutions; to find the minimal variable domains; to find an optimal, or a good solution by means of an objective or multi-objective function defined in terms of certain variables.

2.1 Notation and definitions

We will summarize the notation that is used in this paper.

Generic: The number of variables in a CSP will be denoted by n. The domain of the variable x_i is denoted by D_i. The disjunctive constraints are denoted by c with an index, for example, c_1, c_i, c_k, and the atomic constraints from a disjunctive constraint c_i are denoted by $c_{ip} : p \in \{1..t\}$. The *arity* of a constraint is the number of variables that the constraint involves, so that a non-binary constraint involves any number of variables. When referring to a non-binary CSP, we mean a CSP where some or all of the constraints have an arity of more than 2. Also, all disjunctive constraints have t atomic constraints and all atomic constraints have the maximum arity n.

Variables: To represent variables we use x with an index, for example, x_1, x_i, x_n.

Domains: The continuous domain of the variable x_i is denoted by $D_i = [l_i, u_i]$, so that the domain length of the variable x_i is $d_i = u_i - l_i$.

Constraints: Traditionally, constraints are considered *additive*, that is, the order of imposition of constraints does not matter. All that matters is that the conjunction of constraints be satisfied [1]. Our framework internally manages the constraints in an appropriate order with the objective of reducing the temporal and spatial complexity.

Let $X = x_1, ..., x_n$ be a set of real-valued variables. Let α, β be linear polynomials (i.e. polynomials of degree one) over X. A *linear relation* over X is an expression of the form $\alpha r \beta$ where $r \in \{<, \leq, =, \neq, \geq, >\}$. Particularly, a *linear disequation* over X is an expression of the form $\alpha \neq \beta$ and a *linear equality* over X is an expression of the form $\alpha = \beta$. In accordance with previous definitions, the constraints that we are going to manage are linear relations of the form:

$$Inequalities : \sum_{i=1}^{n} p_i x_i \leq b \tag{1}$$

$$Disequations : \sum_{i=1}^{n} p_i x_i \neq b \tag{2}$$

$$Non-linear\ \ Disequations : F(x) \neq b \tag{3}$$

where x_i are variables ranging over continuous intervals and $F(x)$ is a non-linear function. Equalities can be written as conjunctions of two inequalities, using the above constraints. Similarly, strict inequalities can be written as the conjunction of an inequality and a disequation. Thus, we can manage all possible relations in $\{<, \leq, =, \neq, \geq, >\}$.

These expressions are examples that DPOLYSA can manage:

$(2x_1 - 3x_2 - 5x_3 + x_4 \leq 4)$, $(4x_2^4 + 2x_3 - 2x_5^3 \neq 4)$, $(x_1 + 4x_2 + 5x_3 + 4x_4 < 4)$, $((2x_1 - 3x_2 \leq 4) \vee (x_3 + x_4 \leq 5) \vee (3\sqrt[3]{x_1} + 2x_2^3 - x_4 \neq 5))$

The first and second constraints are managed directly by DPOLYSA, the third constraint is transformed into two constraints:

$(x_1 + 4x_2 + 5x_3 + 4x_4 \leq 4) \wedge (x_1 + 4x_2 + 5x_3 + 4x_4 \neq 4)$

The last constraint is a disjunctive constraint with 3 atomic constraints. Thus, the solution must satisfy one of them.

The following tractable formalisms can be trivially expressed in order to be managed by our proposal DPOLYSA.

Definition 1 *A Horn constraint [7] is a disjunction* $c_i = c_{i_1} \vee c_{i_2} \vee, ..., \vee c_{i_t}$ *where each* $c_{i_k}, k = 1, ..., t$ *is a weak linear inequality or a linear disequation, and the number of inequalities among* $c_{i_1}, ..., c_{i_n}$ *does not exceed one. If there are no inequalities, then a Horn constraint is called negative. Otherwise it is called positive. Horn constraints of the form* $c_{i_1} \vee \cdots \vee c_{i_t}$ *with* $t \geq 2$ *are called disjunctive.*

Example. The following are examples of Horn constraints:

$$x_1 + x_2 - 2x_3 \leq 6, x_1 - 3x_3 + x_4 \neq 3,$$
$$2x_1 - x_3 - x_4 \leq 3 \vee 2x_1 - x_2 + x_4 \neq 4 \vee x_3 - 2x_5 + x_6 \neq 8,$$
$$x_1 - x_2 - x_3 \neq 3 \vee -x_1 - 2x_3 - 4x_4 \neq 8,$$

The first and the third constraints are positive, while the second and the fourth are negative. The third and fourth constraints are disjunctive.

Definition 2 *Let* $r \in \{\leq, \geq, \neq\}$. *A Koubarakis formula [6] is a formula of either of the two forms: (1)* $(x - y)rc$ *or (2)* xrc.

Definition 3 *A simple temporal constraint [2] is a formula of the form:* $c \leq (x - y) \leq d$.

Definition 4 *A simple metric constraint [5] is a formula of the form:* $-cr_1(x - y)r_2d$, *where* $r_1, r_2 \in \{<, \leq\}$.

Definition 5 *A CPA/single interval formula [9] is a formula of one of the following two forms: (1)* $cr_1(x - y)r_2d$ *; or (2)* xry, *where* $r \in \{<, \leq, =, \neq, \geq, >\}$ *and* $r_1, r_2 \in \{<, \leq\}$.

Definition 6 *A TG-II formula [3] is a formula of one of the following forms: (1)* $c \leq x \leq d$, *(2)* $c \leq x - y \leq d$ *or (3)* xry, *where* $r \in \{<, \leq, =, \neq, \geq, >\}$.

Following, we present some definitions that are applied in the paper.

Definition 7 *Given two points $x, y \in R$, a convex combination of x and y is any point of the form $z = \lambda x + (1-\lambda)y$, where $0 \le \lambda \le 1$. A set $S \in R$ is convex if and only if it contains all convex combinations of all pairs of points $x, y \in S$.*

Definition 8 *An Extended DLR is a disjunction $c_i = c_{i_1} \vee c_{i_2} \vee, ..., \vee c_{i_t}$ where each $c_{i_k}, k = 1, ..., t$ is a linear inequality, a linear disequation or a non-linear disequation. A negative Extended DLR is an Extended DLR without inequalities. Otherwise it is called positive. Extended DLRs of the form $c_{i_1} \vee \cdots \vee c_{i_t}$ with $t \ge 2$ are called disjunctive. Each c_{i_k} is called atomic constraint.*

Example. The following are examples of Extended DLRs:

$$x_1 + x_2 - 2x_3 \le 6, x_1 - 3x_3 + x_4 \ne 3,$$
$$2x_1 - x_3 - x_4 \le 3 \vee 2x_1 - x_2 + x_4 \le 4 \vee x_3^2 + x_5 + \sqrt[3]{x_6} \ne 12,$$
$$x_1^4 - x_2 - x_3^2 \ne 3 \vee -x_1 - 2x_3^5 - 4x_4 \ne 8,$$

The first and the third constraints are positive while the second and the fourth are negative. The third and four constraints are disjunctive.

Definition 9 *A continuous CSP whose variables are ranged in non unitary domains, that is, each domain $D_i = [l_i, u_i] : l_i < u_i$, then the CSP is called non-single CSP.*

Theorem 1 *A non-single CSP with a finite set of negative Extended DLRs is consistent.*

Proof:(Proof by Contradiction.) Suppose the CSP is not consistent, that is, there is no point inside the polyhedron generated by the cartesian product of the variable domain bounds:

1.- This polyhedron is a hyperplane $S \subseteq R^{n-1}$ of the entire space R^n and a disequation deletes this hyperplane. However this contradicts the fact that CSP is non-single.

2.- Every point inside the polyhedron is deleted by one or more hyperplanes. These hyperplanes represent the negative disequations. However, there exists an infinite number of points, so an infinite number of disequations is necessary to cover all points $p \subset S$. Again this contradicts the fact that there exists a finite number of disequations.

corollary A *non-single CSP* with disjunctive constraints with at least one disequation per disjunction is consistent.

Proof: Without loss of generality, we can select a disequation for each disjunctive constraint. The resultant set of constraints is a set of negative constraints. By Theorem 1, this set of constraints is consistent, so the entire problem is consistent.

3 Specification of DPOLYSA

DPOLYSA is considered to be a CSP solver that manages Extended DLRs. A general scheme of DPOLYSA is presented in Figure 1. Initially, DPOLYSA runs a preprocessing step in which two algorithms are carried out: the *Constraint Selection Algorithm (CSA)* that selects the non-disjunctive problem that is more likely to be consistent and later the *Constraint Ordering Algorithm (COA)* that classifies the resultant constraints in order to study the more restrictive constraints in first place. Then, using the resultant ordered and non-disjunctive problem, POLYSA [11] carries out the consistency study as a classic CSP solver.

3.1 Preprocessing Step

Solving disjunctive constraint problems requires considering an exponential number of non-disjunctive problems. For example, if the problem has disjunctive constraints composed by two atomic constraints, the number of non-disjunctive problems is 2^k, where k is the number of disjunctive constraints.

Here, we propose CSA that obtains the non-disjunctive problem that is more likely to satisfy the problem. This algorithm can be compared with the sampling from a finite population in which there is a population, and a sample is chosen to represent this population. In this context, the population is the convex hull of all solutions generated by means of the Cartesian Product of variable domain bounds. This convex hull may be represented by a polyhedron with n dimensions and 2^n vertices. However, the sample that the heuristic technique chooses is composed by n^2 items (vertices of the complete polyhedron)[1]. These items are well distributed in order to represent the entire population.

With the selected sample of items (n^2), CSA studies how many items v_{ij} : $v_{ij} \leq n^2$ satisfy each atomic constraint c_{ij} . Thus, each atomic constraint is labelled $c_{ij}(p_{ij})$, where $p_{ij} = v_{ij}/n^2$ represents the probability that c_{ij} satisfies the entire problem. Thus, CSA classifies the atomic constraints in decreasing order and selects the atomic constraint with the highest p_{ij} for each disjunctive constraint.

As we remarked in the preliminaries, constraints are considered *additive*, that is, the order in which the constraints are studied does not make any difference [1]. However, the constraint ordering algorithm (COA) carries out an internal ordering of the constraints. If some constraints are more restricted than others, these constraints are studied first in order to reduce the resultant problem. Thus, the remaining constraints are more likely to be redundant. However, if the remaining ones are not redundant, they generate less new vertices, so the temporal complexity is significantly reduced.

COA classifies the atomic constraints in ascending order with respect the labels p_{ij}. Therefore, the preprocessing step has translated the disjunctive non-binary CSP into a non-disjunctive and ordered CSP in order to be studied by the CSP solver.

Example: Let's take a problem with three variables ($n = 3$), three disjunctive constraints ($k = 3$) with two atomic constraints per disjunction ($t = 2$):

$$c_1 : c_{11} \vee c_{12}$$

[1]The heuristic selects n^2 items if $n > 3$, and 2^n items, otherwise

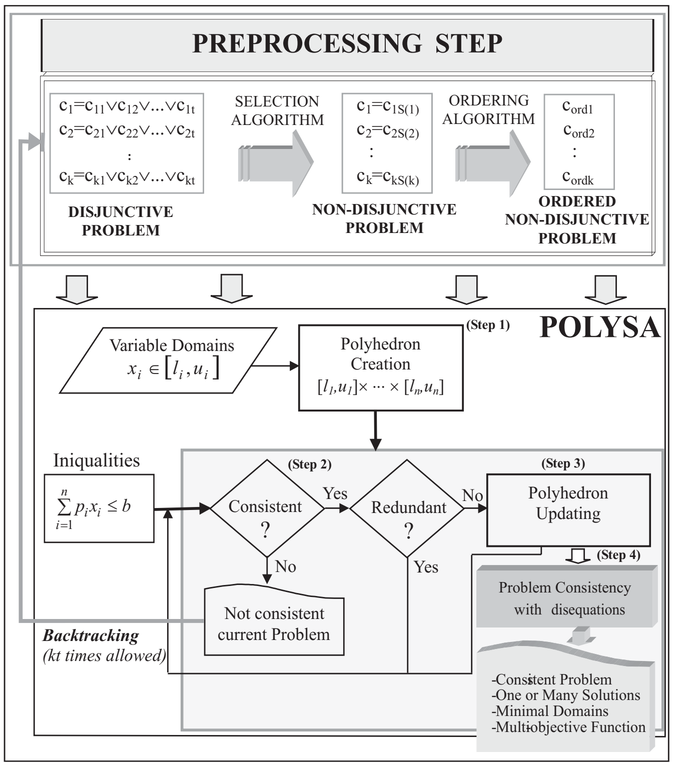

Figure 1: General Scheme of the Disjunctive Polyhedron Search Algorithm

$$c_2 : c_{21} \vee c_{22}$$
$$c_3 : c_{31} \vee c_{32}$$

The algorithm checks how many items (from a given sample: 8 items) satisfy each atomic non-binary constraint and orders them afterwards. Let's assume the following results:

$$v_{11} = 2,\ v_{12} = 6;\ \ v_{21} = 7,\ v_{22} = 4;\ \ v_{31} = 0,\ v_{32} = 7$$
$$p_{11} = 2/8 = 0.25, p_{12} = 0.75, p_{21} = 0.87,\ p_{22} = 0.5, p_{31} = 0, p_{32} = 7/8 = 0.87$$

$$\begin{array}{lcl} c_1 : c_{11}(0.25) \vee c_{12}(0.75) & & c_1 : c_{12}(0.75) \vee c_{11}(0.25) \\ c_2 : c_{21}(0.87) \vee c_{22}(0.50) & \xrightarrow[ordering]{} & c_2 : c_{21}(0.87) \vee c_{22}(0.50) \\ c_3 : c_{31}(0.00) \vee c_{32}(0.87) & & c_3 : c_{32}(0.87) \vee c_{31}(0.00) \end{array}$$

The selected constraints are: (c_{12}, c_{21}, c_{32}), so DPOLYSA will run the corresponding non-disjunctive problem. (See Figure 2)

3.2 DPOLYSA: CSP solver

When the preprocessing step has been carried out, DPOLYSA runs the CSP solver that studies the resulting non-disjunctive and ordered problem. This

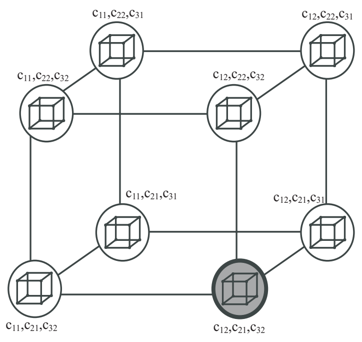

Figure 2: The process of translation process into non-disjunctive problems

CSP solver is called *POLYSA* [11]. The main steps of POLYSA are shown in Figure 1. POLYSA generates an initial polyhedron (step 1) with $2n^3$ vertices created by means of the Cartesian Product of the variable domain bounds ($D_1 \times D_2 \times ... \times D_n$), but randomly selects the vertices so that each polyhedron face maintains n^2 vertices that have not ever been selected by any other adjacent face. POLYSA manages the inequalities and disequations in two different steps. For each inequality, POLYSA carries out the consistency check (step 2). If the inequality is non consistent, POLYSA returns 'not consistent current problem' and DPOLYSA backtracks to the preprocessing step in order to select a new non-disjunctive problem. Otherwise, POLYSA determines whether the inequality is not redundant, and updates the polyhedron (step 3), (i.e.) DPOLYSA eliminates the not consistent vertices and creates the new ones. Finally, when all inequalities have been studied, DPOLYSA studies the consistency with the disequations (step 4). Therefore, the solutions to CSP are the all vertices, as well as all the convex combinations between any two vertices that satisfy all disequations.

It must be taken into account that when the current non-disjunctive problem is not consistent, POLYSA finishes its execution and DPOLYSA backtracks to the preprocessing step in order to select the following best set of non-disjunctive constraints. In the worst case, DPOLYSA backtracks kt times, if the problem is not consistent (where k is the number of disjunctive constraints and t is the number of atomic constraints per disjunction). If DPOLYSA did not run a preprocessing step, then it would be necessary to check t^k non-disjunctive problems in order to study all possibilities. However, experimental result show us that kt backtracking steps was enough for us to study correctly 95% of the problems.

Finally, DPOLYSA can obtain some important results such as:

- The problem consistency: if the polyhedron is not empty.
- One or many problem solutions: solutions to CSP are all vertices and all convex combinations between any two vertices satisfying all disequations.

- The minimal variable domains: these domains are updated (reduced) when each inequality is studied.
- The vertex of the polyhedron that minimizes or maximizes some objective or multi-objective function: the objective function solution is an extreme point of the resultant polyhedron.

It must be taken into account that the DPOLYSA might fail due to the fact that the polyhedron generated by the heuristic does not maintain all the vertices of the complete polyhedron.

Theorem 2 *POLYSA is correct* $\forall n \in N$

Proof: POLYSA is correct $\forall n \in N$ because the resulting polyhedron is convex and is a subset of the resulting convex polyhedron obtained by the complete algorithm HSA [10]. So, we can conclude that POLYSA is correct $\forall n \in N$.

Proposition 3 *For* $n < 12$*, POLYSA is complete*

Proof: POLYSA generates $2n^3$ vertices; when $n < 12$, this number is greater than 2^n. Hence the algorithm is complete since all vertices are covered.

4 Analysis of DPOLYSA

DPOLYSA spatial cost is determined by the number of vertices generated. Initially, in the preprocessing step, DPOLYSA studies the consistency of the n^2 items with the atomic constraints, where n is the number of variables, so the spatial cost is $O(n^2)$. Then, DPOLYSA generates $2n^3$ vertices. For each inequality (step 2), DPOLYSA might generate n new vertices and eliminate only one. Thus, the number of polyhedron vertices is $2n^3 + k(n-1)$ where k is the number of disjunctive constraints. Therefore, the spatial cost is $O(n^3)$.

The temporal cost can be divided into five steps: Preprocessing step, initialization, consistency check with inequalities, updating and consistency check with disequations. In the preprocessing step, CSA checks the consistency of the sample with each atomic constraint, so the temporal cost is $O(ktn^2)$. The atomic constraint ordering in the disjunctive constraints is carried out in $O(ktlog(t))$. COA only classifies the selected set of k atomic constraints in decreasing order. Thus, the temporal cost of the preprocessing step is $O(ktn^2)$, where t is the maximum number of atomic constraints in a disjunctive constraint. The initialization cost (step 1) is $O(n^3)$ because the algorithm only generates $2n^3$ vertices. For each inequality (step 2), the consistency check cost depends linearly on the number of polyhedron vertices, but not on the variable domains. Thus, the temporal cost is $O(n^3)$. Finally, at the worst case the cost of updating (step 3) and the consistency check with disequations depends, on the number of vertices, that is $O(n^3)$. Thus, the temporal cost is: $O(ktn^2) + O(n^3) + kkt \cdot (O(n^3) + O(n^3)) + O(n^3) \Longrightarrow O(k^2tn^3)$. Note that, in practice, this complexity is smaller because the heuristic technique statistically obtains the most appropriate non-disjunctive problem at the preprocessing step, so it is not necessary to try the n allowed possibilities.

5 Evaluation of DPOLYSA

In this section, we compare the performance of DPOLYSA with some of the CSP solvers. We have selected Forward-checking (FC) and Real Full Look-ahead (RFLA)[2] because they are the most appropriate techniques that can manage this CSP typology . We have used a PIII-800 with 256 Mb. of memory and Windows NT operating system.

The random generated problems depended on five parameters $< n, c_{\leq}, c_{\neq}, d, t >$, where n was the number of variables, $c_{\leq}$ the number of disjunctive inequalities, $c_{\neq}$ the number of disequations, d the length of variable domains and 't' the number of atomic inequalities per disjunction. To evaluate the behavior of the algorithms with the selected domains, we fixed the domain length to their maximum values in Figures 4, 6 and 7. In Figures 3 and 5, we fixed the maximum bounds of the variable domains. These random domains could be lower. The problems were randomly generated by modifying these parameters. We considered all constraints with non-null coefficients, that is $p_i \neq 0 \ \forall i = 1...n$. Thus, each of the graphs shown sets four of the parameters and varies the other one in order to evaluate the algorithm performance when this parameter increases. We tested 100 test cases for each type of problem and each value of the variable parameter, and we present the mean CPU time for each of the techniques. Five graphs are shown below. (Figures 3, 4, 5, 6, 7) which correspond to the five significant parameters. Each graph summarizes the Mean CPU time for each technique. Here, for unsolved problems in 200 seconds, we assigned a 200-second run-time. Therefore, this graph contains a horizontal asymptote in $time = 200$.

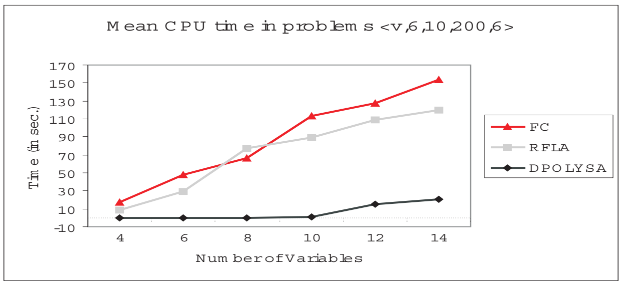

Figure 3: Mean CPU Time when the number of variables increases

In Figure 3, the number of disjunctive inequalities, the number of disequations, the maximum bounds of the variable domains and the number of atomic inequalities were set $< n, 6, 20, 200, 6 >$, and the number of variables was increased from 4 to 14. The variable domains were randomly chosen between

[2] Forward-checking and Real Full Look-ahead were obtained from CON'FLEX, which is a C++ solver that can handle constraint problems with continuous variables and disjunctive constraints. It can be found in: http://www-bia.inra.fr/T/conflex/Logiciels/adressesConflex.html.

$[-200, 200]$, so the variables took different domains: $[l, u] \subseteq [-200, 200]$. The graph shows a global view of the behavior of the algorithms. The mean CPU time in FC and RFLA increased faster than DPOLYSA, which only increased its temporal complexity polynomially. When the unsolved problems were set to time=200, and the others maintained their real time cost, we observed that FC was worse than RFLA. However, DPOLYSA always had a better behavior and was able to solve all problems satisfactorily.

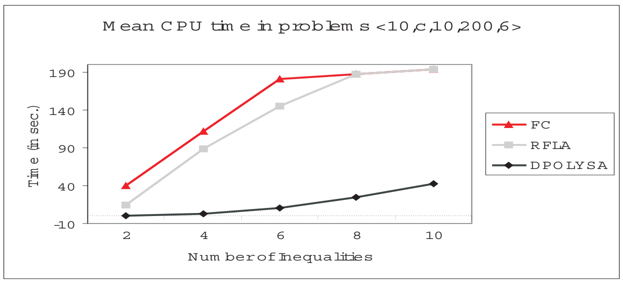

Figure 4: Mean CPU Time when the number of disjunctive inequalities increases

In Figure 4, the number of variables, the number of disequations, the domain length and the number of atomic inequalities were set $< 10, c, 40, 200, 6 >$, and the number of random disjunctive inequalities ranged from 2 to 10. In this case, the domain length was fixed, so that all variable domains were $[-200, 200]$.

The graph shows that the mean CPU times in FC and RFLA increased exponentially and were near the horizontal asymptote for problems with 10 disjunctive inequalities. However, DPOLYSA only increased its temporal complexity polynomially. The number of unsolved problems increased in FC and RFLA much more than in DPOLYSA. Also, the behavior of FC and RFLA was worse in fixed domains than in random domains. This can be seen in Figure 3 when $v = 10$ and Figure 4 when $c = 6$. In both cases the problem is $< 10, 6, 40, 200, 6 >$ and the mean CPU time is different in both graphs.

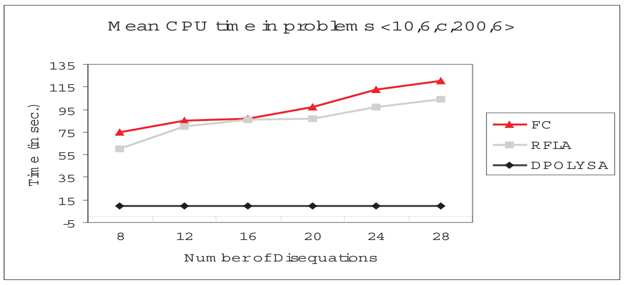

Figure 5: Mean CPU Time when the number of disequations increases

In Figure 5, the number of variables, the number of disjunctive inequalities, the domain length and the number of atomic inequalities were set $< 10, 6, c, 200, 6 >$, and the number of random disequations ranged from 6 to 26. The variable domains were randomly chosen between $[-200, 200]$. The graph shows that the behavior of FC and RFLA got worse when the number of disequations increased. DPOLYSA did not increase its temporal complexity due to the fact that it carried out the consistency check of the disequations in low complexity. The number of unsolved problems was very high for both FC and RFLA, while DPOLYSA had a good behavior. Note that DPOLYSA was proved with an amount of 10^5 disequations and it solved them only in a few seconds (< 3 sc.)

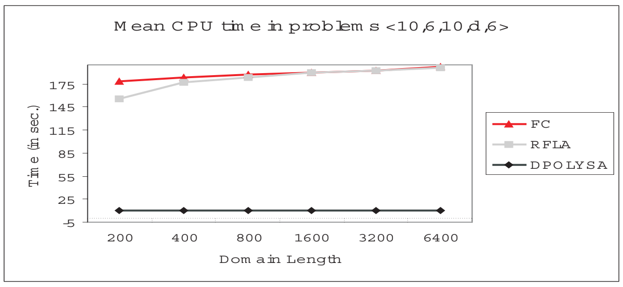

Figure 6: Mean CPU Time when the number of domain length increases

In Figure 6, the number of variables, disjunctive inequalities, disequations and atomic inequalities were set $< 10, 6, 10, d, 6 >$, and the domain length was increased from $[-200, 200]$ to $[-6400, 6400]$.

The graph shows that the behavior of FC and RFLA got worse in all domain length. DPOLYSA had a constant temporal complexity because this complexity is independent from the domain length. The number of unsolved problems was very high for both FC and RFLA, while DPOLYSA had a good behavior.

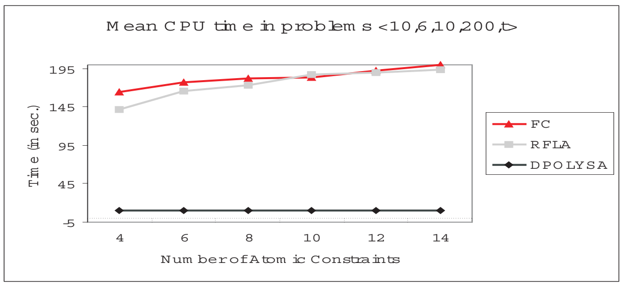

Figure 7: Mean CPU Time when the number of atomic constraints increases

In Figure 7, the number of variables, disjunctive inequalities, disequations, and the domain length were set $< 10, 6, 10, 200, t >$, and the atomic inequalities were increased from 4 to 14. In this case, the domain length was fixed to $[-200, 200]$. To study the behavior of the algorithms when the number of atomic inequalities increased, we chose $t-1$ non-consistent atomic inequalities and only one inequality atomic constraint. That is, if the number of atomic inequalities was 8, the random constraint generator generated 7 non-consistent atomic inequalities and 1 consistent constraint. Thus, we could observe the behavior of the algorithm when the number of atomic inequalities increased. FC and RFLA had worse behavior than DPOLYSA. DPOLYSA makes a preprocessing step in which it selects the most appropriate non-disjunctive problem. Also, this preprocessing step is made in polynomial time, so the temporal cost is very low.

Remark 1 . *These tests were stopped at 200 seconds because it is the limit in which RFLA and FC had solved most of the problems. Between 200 and 1000 seconds, less than 3% of problems were solved by FC and RFLA.*

We present a comparison between RFLA and FC using the proposed preprocessing step (RFLA with preprocessing and FC with preprocessing) and not using it (RFLA without Preprocessing and FC without Preprocessing) in Table 1. The random generated problems had the following properties: the number of variables, the number of disequations, the variable domain length and the number of atomic inequalities were set $< 5, c, 5, 10, 2 >$ and the number of disjunctive inequalities were increased from 5 to 25. It can be observed that the preprocessing step reduced the number of constraint tests in both algorithms. The difference between FC without preprocessing and FC with preprocessing would increase if the number of atomic constraints were higher than 2.

	Number of disjunctive constraints				
Algorithm	5	10	15	20	25
FC without Preprocessing	24	44	64	84	104
FC with Preprocessing	14	24	34	44	54
RFLA without Preprocessing	18.2	56.1	77.8	162.6	504.1
RFLA with Preprocessing	10.6	38.8	41.3	85.1	262.2

Table 1: Average number of constraint tests in problems $< 5, c, 2, 10, 2 >$.

6 Conclusions

In this paper, we have extended the class of DLRs to include disjunctions with an arbitrary number of linear inequalities, linear disequations and non-linear disequations. This new class of constraints called *Extended DLRs* subsumes several other classes of constraints. Moreover, our objective is not only to decide consistency, but also to obtain one or several solutions and to obtain the minimal domain of the variables. To achieve this objective, we have proposed an algorithm called DPOLYSA that solves the class of Extended DLRs as a non-binary disjunctive CSP solver. This proposal carries out two preprocessing algorithms. CSA translates the disjunctive problem into a non-disjunctive one and COA classifies the atomic constraints in an appropriate order. Then, POLYSA runs the resulting non-disjunctive problem.

References

[1] R. Bartk, 'Constraint programming: In pursuit of the holy grail', *in Proceedings of WDS99 (invited lecture), Prague, June*, (1999).

[2] R. Dechter, I. Meiri, and J. Pearl, 'Temporal constraint network', *Artificial Intelligence*, **49**, 61–95, (1991).

[3] A. Gerevini, L. Schubert, and S. Schaeffer, 'Temporal reasoning in time graph i-ii', *SIGART Bull 4*, **3**, 21–25, (1993).

[4] P. Jonsson and Backstrom C., 'A linear programming approach to temporal reasoning', *In Proceedings of AAAI-96*, 1235–241, (1996).

[5] H. Kautz and P. Ladkin, 'Integrating metric and temporal qualitative temporal reasoning', *In Proc. 9th National Conference Artificial Intelligence (AAAI-91)*, 241–246, (1991).

[6] M. Koubarakis, 'Dense time and temporal constraints with ¡¿', *In Proc. 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, 24–35, (1992).

[7] M. Koubarakis, 'Tractable disjunction of linear constraints', *In Proc. 2nd International Conference on Principles and Practice of Constraint Programming (CP-96)*, 297–307, (1999).

[8] J.L. Lassez and K. McAloon, 'A canonical form for generalizad linear constraints', *In Advanced Seminar on Foundations of Innovative Software Development*, 19–27, (1989).

[9] I. Meiri, 'Combining qualitative and quantitative constraints in temporal reasoning', *Artificial Intelligence*, **87**, 343–385, (1996).

[10] M.A. Salido and F. Barber, 'An incremental and non-binary CSP solver: The Hyperpolyhedron Search Algorithm', *In Proceedings of Constraint Programming (CP-2001), LNCS 2239*, 799–780, (2001).

[11] M.A. Salido and F. Barber, 'POLYSA: A polinomial algorithm for non-binary constraint satisfaction problems with <= and <>', *In Proceeding of EPIA-2001 Worshop on Constraint Satisfaction and Operation Research (CSOR01)*, 99–113, (2001).

[12] E. Sontag, 'Real addition and the polynomial time hierarchy', *Information Processing Letter*, **20**, 115–120, (1985).