POLYSA: A Polynomial Algorithm for Non-binary Constraint Satisfaction Problems with ≤ and ≠

Miguel A. Salido, Federico Barber

Dpto. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia, Camino de Vera s/n 46071
Valencia, Spain
{msalido, fbarber}@dsic.upv.es

Abstract. Nowadays, many real problems can be naturally modelled as Constraint Satisfaction Problems (CSPs). It is well known that any non-binary CSP can be transformed into an equivalent binary one, using some of the current techniques. However, this transformation may not be practical in problems with certain properties. Therefore, it is necessary to manage these non-binary constraints directly. In this paper, we propose a heuristic called "*POLYSA*" that solves non-binary constraint satisfaction problems in a natural way as an incremental and non-binary CSP solver. This non-binary CSP solver carries out the search through a polyhedron that maintains in its vertices those solutions that satisfy all non-binary constraints.

1. Introduction

Nowadays, many problems in the 'real world' can be efficiently modelled as constraint satisfaction problems and solved using constraint programming techniques. These include problems from fields such as artificial intelligence, operational research, databases, expert systems, etc. Most of these problems can be modelled naturally as non-binary constraint satisfaction problems. Modelling a problem with non-binary constraints has several advantages. It facilitates the expression of the problem, enables more powerful constraint propagation as more global information is available, etc. It is well known that any non-binary CSP can be solved directly by means of non-binary CSPs solvers or transformed into a binary one [10]. However, this transformation has several drawbacks:

- The translation process generates new variables, so it produces a significant increase in the problems size, causing extra memory requirements for algorithms. In some cases, solving the binary formulation can be very inefficient [1] [3] [8].
- In any case, this forced binarization generates unnatural formulations, which cause extra difficulties for constraint solver interfaces with human users [2].

In this paper, we propose an algorithm called "***Poly****hedron* ***S****earch* ***A****lgorithm* " (*POLYSA*) that manages non-binary CSPs in a natural way as an incremental and non-binary CSP solver. This algorithm is based on HSA [11][12] which is the

complete algorithm because it maintains all the vertices generated by the Cartesian Product of the variable domain bounds. Although HSA obtains good experimental results, its computational complexity is $O(2^n)$, so the use of heuristics is necessary to solve these problems in polynomial time. Therefore, we present *POLYSA* which shares the same philosophy as HSA, (i.e) the vertices are generated by the Cartesian Product of some variable domain bounds. However its computational complexity is reduced to $O(n^3)$, although *POLYSA* is not a complete algorithm.

POLYSA efficiently manages non-binary CSPs with many variables, many constraints and very large domains. This proposal overcomes some of the weaknesses of other typical techniques, like *Forward-Checking* and *Real Full Look-Ahead*, since its complexity does not change when the domain size and the number of disequational constraints increase. Moreover, we can manage constraints (inequalities $\leq$) and (disequations $\neq$) that can be inserted incrementally and dynamically into the problem without having to solve the whole problem again. Thus, we can manage non-binary dynamic constraints which are very frequent in many real applications.

This paper is structured as follows: First, we present the preliminaries and the constraint typology that *POLYSA* can manage. Then, we present the algorithm and evaluate its behaviour with some of the current techniques. Finally, we present the conclusions and future work proposals.

2. Preliminaries

Briefly, a constraint satisfaction problem (CSP) that *POLYSA* manages consists of:

- A set of *variables* $X = \{x_1,..., x_n\}$.
- For each variable $x_i \in X$ there is a domain of possible values D_i.
- A set of *constraints* $C = \{c_1,..., c_p\}$ restricting the values that the variables can simultaneously take.

A solution to a CSP is an assignment of a value from its domain to every variable, such that all constraints are satisfied.

The objective in a CSP may be:

- to determine whether a solution exists
- to find one solution, (with no preference as to which one), many or all solutions
- to find the 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 we will use in this paper.

Generic: The number of variables in a CSP will be denoted by *n*. The domain of the variable x_i will be denoted by D_i. The constraints will be noted by *c*, and all the constraints have the maximum arity *n*. We will always use this notation when analysing the complexities of algorithms.

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

Domains: The domain of the variable x_i will be denoted by $D_i = [l_i, u_i]$, so the domain length of the variable x_i is $u_i - l_i$. It is important to realise that the domain is continuous.

Constraints: Let $X = \{x_1, ..., x_n\}$ be a set of real-valued variables. Let α be a polynomial of degree n (i.e., $\alpha = \sum_{i=1}^{n} p_i x_i$) over *X* and *b* an integer. A *linear relation* over *X* is an expression of the form $\alpha r b$ where $r \in \{<, \leq, =, \neq, \geq, >\}$. A *linear disequation* over *X* is an expression of the form $\alpha \neq b$. A *linear equality* over *X* is an expression of the form $\alpha = b$. The constraints that we are going to manage are linear relations and linear disequations of the form:

$$\text{Inequalities: } \sum_{i=1}^{n} p_i x_i \leq b \quad (\mathbf{1})$$

$$\text{Disequations: } \sum_{i=1}^{n} p_i x_i \neq b \wedge F(x) \neq b : F(x)\ non-linear \quad (\mathbf{2})$$

where x_i are variables ranging over continuous intervals $x_i \in [l_i, u_i]$, b is a real constant, and $n \geq 1$. Using the above constraints, equalities can be written as conjunctions of two inequalities. Similarly, strict inequalities can be written as the conjunction of an inequality and an disequation. Thus, we can manage all possible combinations: $\{<, \leq, =, \neq, \geq, >\}$.

Definiton 1.

Given two points $x,y \in \mathbb{R}^n$, a *convex combination* of *x* and *y* is any point of the form $z = \lambda x + (1-\lambda) y$ where $0 \leq \lambda \leq 1$. A set $S \subseteq \mathbb{R}^n$ is convex iff it contains all convex combinations of all pairs of points $x,y \in S$.

Definition 2.

Let $r \in \{\leq, \geq, \neq\}$. A *Koubarakis* formula [7] is a formula of either of the two forms (1) $(x-y)\ r\ c$ or (2) $x\ r\ c$.

Definition 3.
A *simple temporal constraint* [4] is a formula of the form $c \leq (x\text{-}y) \leq d$.

Definition 4.
A *simple metric constraint* [6] is a formula of the form $-c\ r_1\ (x\text{-}y)\ r_2\ d$ 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) $c\ r_1\ (x\text{-}y)\ r_2\ d$; or (2) $x\ r\ y$ where $r \in \{<, \leq, =, \neq, \geq, >\}$ and $r_1, r_2 \in \{<, \leq\}$.

Definition 6.
A *TG-II* formula [5] is a formula of one of the following forms: (1) $c \leq x \leq d$, (2) $c \leq x\text{-}y \leq d$ or (3) $x\ r\ y$ where $r \in \{<, \leq, =, \neq, \geq, >\}$.

Note that the tractable formalisms defined in Definitions 2-6 can be trivially expressed in order to be managed by our proposal *POLYSA*.

2.2 Constraints

We now give more definitions on constraints and distinguish between binary and non-binary constraints.
The *arity* of a constraint is the number of variables that the constraint involves. A *unary* constraint is a constraint involving one variable. A *binary* constraint is a constraint involving a pair of variables. A non-binary constraint is a constraint involving an arbitrary 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. *POLYSA* is a CSP solver that manages non-binary constraints.

Example.
The following are examples of non-binary constraints:

$(2x_1 - 6x_2 + 3x_3 - 9x_4 \leq 4)$, $(3x_1x_3 - 2x_4^3 - 3x_5 \neq 4)$, $(x_1 - 2x_2 - 4x_3 + x_4 < 4)$

The first and second constraints are managed directly by *POLYSA*, the last constraint is transformed into two constraints:
$(x_1 - 2x_2 - 4x_3 + x_4 < 4) \Rightarrow (x_1 - 2x_2 - 4x_3 + x_4 \leq 4) \wedge (x_1 - 2x_2 - 4x_3 + x_4 \neq 4)$

3. Specification of the *Polyhedron Search Algorithm*

Initially, *POLYSA* can be considered as a classic CSP solver, where there is a static set of constraints that the solution must satisfy. The main steps are shown in

Fig1. *POLYSA* generates an initial polyhedron (step 1) with $2n^3$ vertices created by means of the Cartesian product of the variable domain bounds ($D_1 \times D_2 \times \cdots \times D_n$), but randomly selecting the vertices such that each polyhedron face maintains n^2 vertices that have not ever been selected by any other adjacent face.

For each input (≤) constraint, *POLYSA* carries out the consistency check (step 2). If the (≤) constraint is not consistent, *POLYSA* returns *not consistent problem*; otherwise, *POLYSA* determines whether the (≤) constraint is not redundant, and updates the polyhedron (step 3), (i.e.) *POLYSA* eliminates the not consistent vertices and creates the new ones. When all static (≤) constraints are studied, MAS checks the consistency with the (≠) constraints (step 4).

Thus, solutions to CSP are all vertices and all convex combinations between any two vertices satisfying all disequational constraints.

Finally, *POLYSA* can obtain some important results such as: (1) the problem consistency; (2) one or many problem solutions; (3) the new variable domains; and (4) the vertex of the polyhedron that minimises or maximises some objective or multiobjective function.

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

Furthermore, when *POLYSA* finishes its static behaviour (classic CSP solver), new constraints can be incrementally inserted into the problem, and *POLYSA* studies the consistency check as an incremental CSP solver.

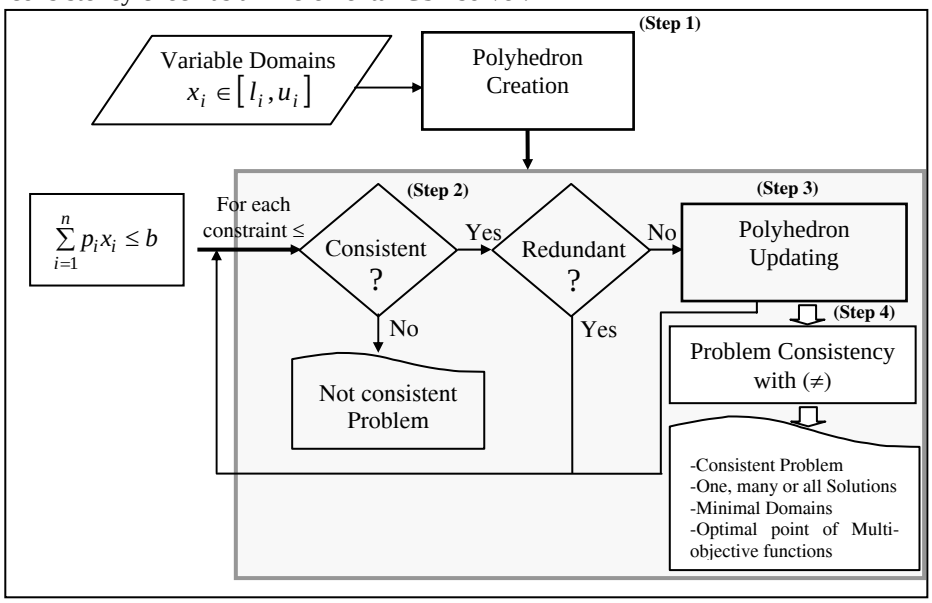


Fig. 1. General Scheme of the Polyhedron Search Algorithm.

3.1 The Polyhedron Search Algorithm

The main goal of *POLYSA* is to solve a problem as an incremental and non-binary CSP solver. The main specification of *POLYSA* is defined in Fig. 4. This specification includes three procedures:

- *Polyhedron_Creation (Dimension, domains)* (Fig 2) is a module that generates the polyhedron vertices by means of the Cartesian Product of the variable domains obtaining some of the polyhedron vertices:
  $v_1 = (l_1, l_2, \cdots, l_n), \cdots, v_i = (l_1, l_2, \cdots, l_j, u_{j+1}, \cdots, u_n), \cdots, v_n = (u_1, u_2, \cdots, u_n)$

```
Polyhedron_creation (Dimension, Domains)
{
For i=0 to 2*Dimensions do:              // For each face do the following
        {
        For j=0 to Dimensions*Dimensions do:
        {
                Make(NewVertex);         // New empty vertex is generated
                NewVertex ← Assigns a new unrepeated vertex in the current face
                ListV ∪ NewVertex;
        }
}
Return ListV ;  // the module returns a list with 2*Dimension^3 vertices.
}
```

Fig. 2. Polyhedron_creation (step 1)

- *Satisfy(C_i, v_i)* is a module that determines whether the vertex v_i satisfies the constraint C_i. Thus, this function only returns true if the result is $\leq b$, when the variables $(x_1, x_2, \cdots, x_n)$ are set to values $v_i = (v_1, v_2, \cdots, v_n)$.

- *Polyhedron_Updation* (v', L_{yes}, L_{no}) (Fig 3) is a module that updates the polyhedron eliminating all inconsistent vertices (L_{no} are the vertices that do not satisfy the constraint) and includes the new vertices generated by the intersection between arcs that contain a consistent extreme (vertex) and the other v'.

```
Polyhedron_Updation ( v' , L_yes , L_no )
{
For each adjacent vertex v of v' do:
        POLYSA obtains the straight line l̄ that unites both v' and v points.
        POLYSA intersects l̄ with the polyhedron obtaining the new point v''
        L_yes ← L_yes ∪ v''
return L_yes ;
}
```

Fig. 3. Polyhedron_Updation (step 3)

```
POLYSA (Dimension, Domains, Constraints≤ ,Constraints≠ , Solutions)
{
Step 1  ListV  ← Polyhedron_Creation (Dimension, Domains, Constraints≤);
L_yes ← φ; L_no ← φ;
Step 2..For each C_i ∈ Constraints≤  do:
  {
  ∀v_i ∈ ListV do:
          {
          If Satisfy(C_i, v_i) then:          // checks if the v_i satisfies the constraint C_i
                  L_yes ← L_yes ∪ {v_i};          // C_i  is consistent with the system
          else  L_no ← L_no ∪ {v_i};
          }
  If  L_yes = φ ⇒ STOP ;          // C_i is not consistent with the system
  If  L_no = φ ⇒ " C_i is consistent and redundant ";
      else
Step 3  ∀v' ∈ L_no  Polyhedron_Updation ( v' , L_yes, L_no );
  }
Step 4 Satisfaction(Constraints≠,  L_yes , Solutions);          // see section 3.2
return output;  // OFHH returns the consistency check, and some extreme solutions

}
```

Fig. 4. Polyhedron Search Algorithm.

Theorem 1*: POLYSA* is correct $\forall n \in \mathrm{N}$.

Proof: *POLYSA* is correct $\forall n \in \mathrm{N}$ because the resulting polyhedron is convex and is a subset of the resulting convex polyhedron obtained by the complete algorithm HSA[12]. So, we can conclude that *POLYSA* is correct $\forall n \in \mathrm{N}$.

Theorem 2*:* For $n < 10$, *POLYSA* is complete.

Proof*:* Suppose by contradiction that *POLYSA* is not complete for $n<10$.
The initial polyhedron which is generated by the Cartesian Product of the variable domain bounds, contains $2n$ faces and 2^n vertices.

POLYSA generates n^2 *unrepeated* vertices in each face. Thus, *POLYSA* generates $2n \cdot n^2 = 2n^3$ vertices. If *POLYSA* is not complete for $n<10$, then there are some vertices that *POLYSA* does not maintain for $n<10$. So $2n^3 < 2^n$ for $n<10$. This is a contradiction, because $2^n < 2n^3$ for $n<10$. So, *POLYSA* is complete for $n<10$.

3.2 Consistency study with constraints ≠

In this section, we present the behavior of *POLYSA* when there are some (≠) constraints. The consistency study of constraints of this type has a linear complexity with the number of (≠) constraints. Following, in Fig. 5, we present an example in $\mathbb{R}^2$ that summarizes the possible situations that might be reached.

- Polyhedron (1) represents the initial polyhedron created by the Cartesian Product of the two variables.
- Polyhedron (2) represents the resulting polyhedron when all the (≤) constraints have been analyzed.

So, when *POLYSA* has checked the problem consistency with the constraints (≤) and it is consistent, *POLYSA* carries out the consistency check of the constraints (≠). Therefore, *POLYSA* maintains in a list, L_{yes} , all the resulting polyhedron vertices. The consistency check with the constraints (≠) and the list, L_{yes} , can generate four different cases:

(a) Some vertices of the list, L_{yes} , are consistent with all the (≠) constraints. In this case, the problem is consistent.

(b) Any vertex of the list 'L_{yes}' is consistent with all the (≠) constraint. However, some convex combinations between two vertices are consistent with the (≠) constraint. In this case, the problem is consistent.

(c) There is a (≤) constraint that reduces the polyhedron to a unique face, and the resultant vertices are not consistent with the (≠) constraints. However, some convex combinations between two vertices are consistent with the constraint (≠). In this case, the problem is consistent.

(d) The last situation occurs when there is a (<) constraint, that is, a (≤) constraint that reduces the polyhedron to a unique face and a (≠) constraint that eliminates this face. Thus, no vertex is consistent with the constraints and no convex combination is consistent. Therefore, the problem is not consistent.

These constraints can be dynamically inserted or deleted from the problem due to the fact that the management of the (≠) constraints has a low computational complexity. When a new (≠) constraint is inserted, the consistent vertices to this point are checked with the new one. It must be taken into account that if at least one vertex is consistent, the whole problem will be consistent. If a non-binary (≠) constraint is deleted from the problem, the algorithm looks for the vertices that have been eliminated by this constraint and determine whether these vertices have also been eliminated by other constraints. The vertices which have been eliminated by this constraint only are returned to the list 'L_{yes}'. If this list is not empty, the problem is consistent. Otherwise, the problem continues to be not consistent.

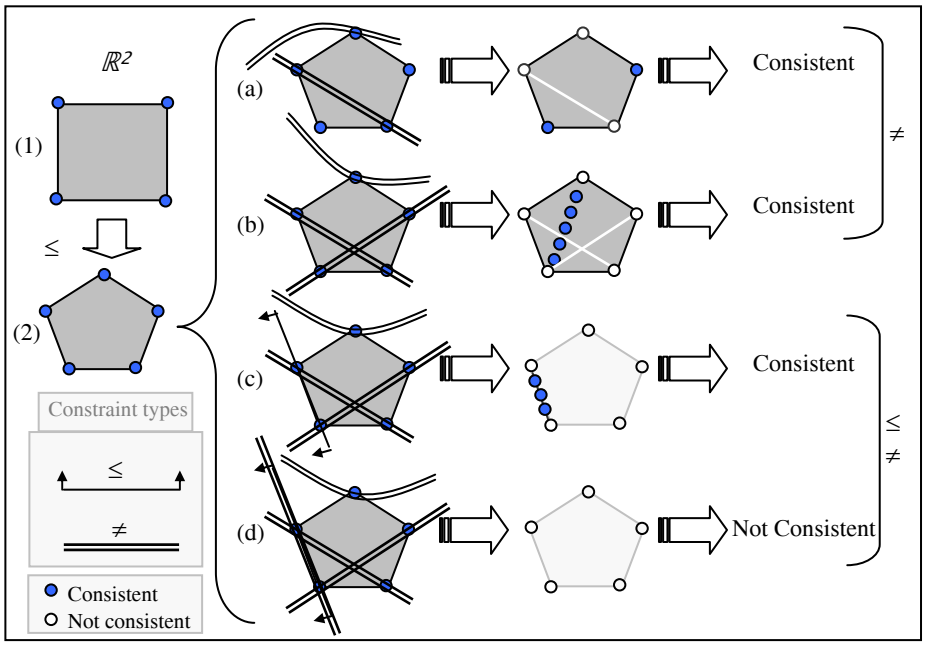


Fig. 5. Different situations in (≠) constraints.

3.3 Graphical Interface

POLYSA allows the user to introduce the necessary parameters in order to randomly or manually run problems. The graphical interface is presented in Fig 6. The upper part of the screen shows the parameters that the user must configure: the number of variables, the number of (≤) constraints, the number of (≠) constraint, the domain length, the number of desired solutions if the problem is consistent, and finally, the number of problems if the user wants the program to generate several random problems. Then, when the parameters are set, the selected problems are randomly generated and solved by *POLYSA*.

However, if the user wants to manually generate a problem, it is possible to introduce the variable domains and the selected constraints in the corresponding parameters.

The problems generated and solved in the lower window of Fig. 6 are shown. This screen shows the selected parameters, the random or manual variable domains and constraints. Also, this screen displays some information about the execution, showing a partial solution and the CPU time for checking each (≤) constraint. Finally, this screen displays the consistency problem and the total CPU time. If the problem is consistent, the desired solutions are shown.

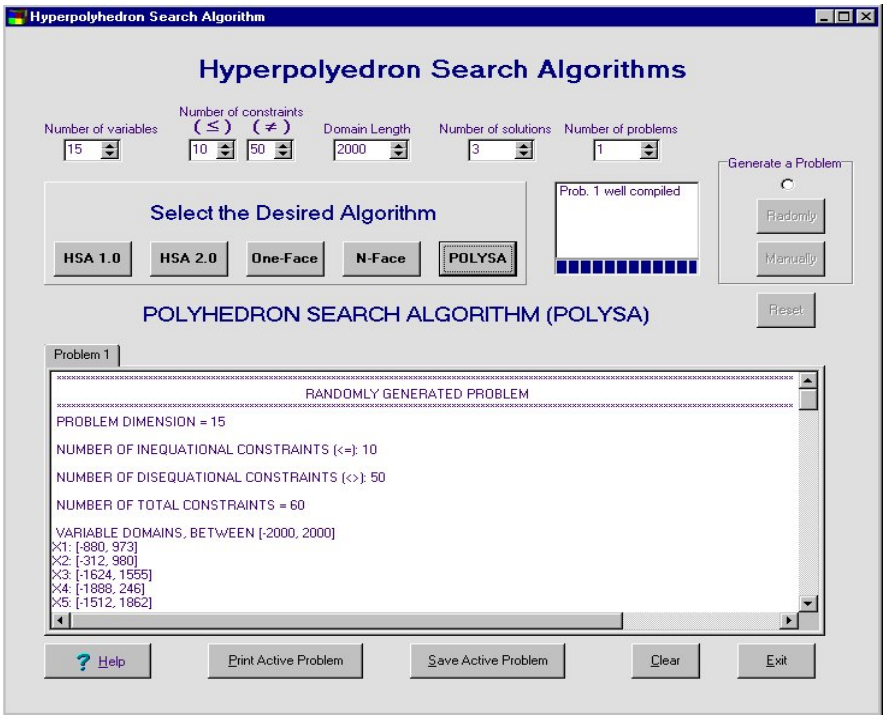


Fig. 6. Graphical Interface

4. Analysis of the Polyhedron Search Algorithm

POLYSA spatial cost is determined by the number of vertices generated. Initially, *POLYSA* generates $2n^3$ vertices, where n is the number of problem variables. For each (≤) constraint (step 2), *POLYSA* 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 constraints. Therefore, the spatial cost is $O(n^3)$. However, HSA spatial cost is $O(2^n)$ due to the fact that the algorithm generates all the polyhedron vertices, and it would be impractical in problems with many variables.

The temporal cost is divided into three steps: initialisation, consistency check and actualisation. The initialisation cost (step 1) is $O(n^3)$ because the algorithm only generates $2n$ vertices. For each (≤) constraint (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, the actualisation cost (step 3) depends on the number of variables $O(n^3)$. Thus, the temporal cost is: $O(n^3) + k * \left(O(n^3) + O(n^3)\right) \Rightarrow O(k \cdot n^3)$.

5. Evaluation of the Polyhedron Search Algorithm

In this section, we compare the performance of *POLYSA* with some of the more current CSP solvers. We have selected Forward-checking (FC) and Real Full Look-ahead (RFLA)[1] because they are the most appropriate techniques that can manage this problem typology . To evaluate this performance, the computer used for the tests was a PIII-800 with 128 Mb. of memory and Windows NT operating system.

The problems generated to evaluate the performance depended on four parameters $<v,c_{\leq},c_{\neq},d>$, where v was the number of variables, $c_{\leq}$ the number of (≤) constraints, $c_{\neq}$ the number of (≠) constraints and d the length of the variable domains. The problems were randomly generated by modifying the parameters and the $c_{\leq}$ and $c_{\neq}$ constraints were constraints of type (1) and (2), respectively, with the coefficients $p_i \neq 0$.

Thus, each of the graphs shown sets three parameters and varies the other in order to evaluate the algorithm performance when this parameter increases.

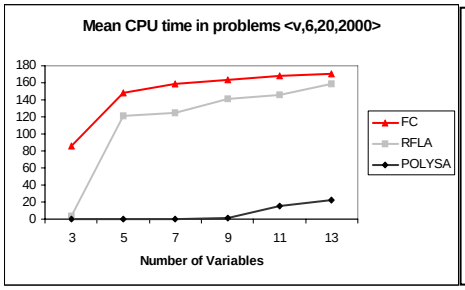
We tested 100 test cases for each problem and the value of the variable parameter, and we present the mean CPU time for each of the techniques.

Following, four sets of graphs are shown. (Figures 7,8,9,10) corresponding to the four significant parameters. Each set summarises the algorithm behaviour from two different perspectives.

(a) Mean CPU time for each technique. It must be taken into account that the *unsolved problems* were assigned a 200-second run-time Thus, this graph contains a horizontal asymptote in time=200.

(b) Number of *unsolved problems*. That is, those problems that were not solved in 200 seconds and problems that did not obtain the desired solution.

In Fig.7, the number of (≤) and (≠) constraints and domain length were set $<v,6,20,2000>$, and the number of variables was increased from 3 to 13.



Number of unsolved problems

	3	5	7	9	11	13
FC	34	64	72	77	78	80
RFLA	0	52	54	63	64	73
POLYSA	0	0	0	0	0	0

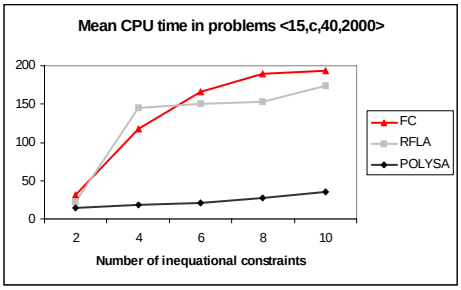
Number of variables

Fig. 7. Temporal Cost in problems $<v,6,20,2000>$.

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

The graph shows a global view of the behaviour of the algorithms. The mean CPU time in FC and RFLA increased exponentially with the number of variables. *POLYSA* 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, as the number of variables increased, *POLYSA* had a better behaviour. The number of unsolved problems increased in FC and RFLA. *POLYSA* was able to solve all problem satisfactorily.

In Fig.8 the number of variables, the number of (≠) constraints and the domain length were set <15, $c_{\leq}$, 40, 2000>, and the number of random (≤) constraints ranged from 2 to 10.



Number of unsolved problems

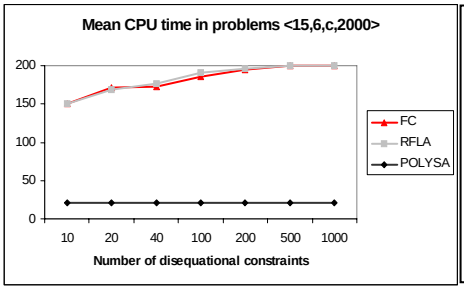
	2	4	6	8	10
FC	14	52	80	94	96
RFLA	10	69	71	77	85
POLYSA	0	0	0	5	10

Number of inequational constraints

Fig. 8. Temporal Cost in problems <15, $c_{\leq}$, 40, 2000>

The graph shows that the mean CPU times in FC and RFLA increased exponentially and were near the horizontal asymptote for problems with 10 (≤) constraint, however, RFLA had a better behaviour than FC. *POLYSA* only increased its temporal complexity polynomial, but when the number of constraints increased, the probability of failure also increased. The number of unsolved problems increased in FC and RFLA much more than in *POLYSA,* which was unable to solve only 10% of the problems with 10 (≤) constraints.

In Fig.9, the number of variables, the number of (≤) constraints and domain length were set <15, 6, $c_{\neq}$, 2000>, and the number of random (≠) constraints ranged from 10 to 1000.



Number of unsolved problems

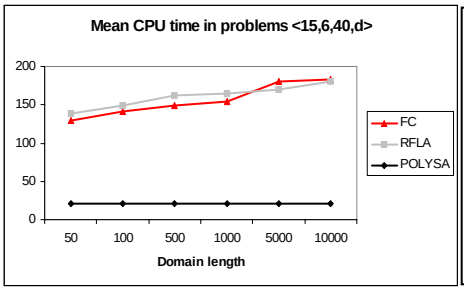
	10	20	40	100	200	10000
FC	75	85	86	92	96	100
RFLA	75	82	86	90	95	100
POLYSA	0	0	0	0	0	0

Number of disequational constraints

Fig.9. Temporal Cost in problems <15, 6, $c_{\neq}$, 2000>

The graph shows that the behavior of FC and RFLA got worse when the number of (≠) constraints increased. *POLYSA* did not increase its temporal complexity due to the fact that it carried out the consistency check of the (≠) constraints in linear complexity. The number of unsolved problems was very high for both FC and RFLA, while *POLYSA* had a good behavior.

In Fig.10 the number of variables and the number of random constraints were set <15, 6, 40, d>, and the domain length was increased from 50 to 10000.



Number of unsolved problems

	50	100	500	1000	5000	10000
FC	62	70	74	76	88	90
RFLA	66	70	78	82	84	90
POLYSA	0	0	0	0	0	0

Domain Length

Fig. 10. Temporal Cost in problems <15, 6, 40, d>.

The graph shows that the behavior of FC and RFLA got worse when the domain length increased. *POLYSA* 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 *POLYSA* had a good behavior.

6. Conclusions and Future Works

In this paper, we have proposed an algorithm called *POLYSA* as an incremental and non-binary CSP solver. This proposal carries out the consistency study through a polyhedron that maintains in its vertices, those values that satisfy all metric temporal constraints. Thus, solutions to CSP are all vertices and all convex combinations between any two vertices that satisfy the disequational constraints.

The computational cost depends polynomial on the number of variables, while other approaches depend exponentially on the number of variables, the number of constraints and the domain size.

Because of the low temporal cost, an unsolved problem can be run several times to reduce the probability of *POLYSA* failing. This technique is carried out by varying the selected vertices that compose the polyhedron.

Currently, we are working on more complete heuristics in order to decrease the probability of unsolved problems. We are working on disjunctive and non-binary

constraint satisfaction problems to be modelled as distributed and incremental CSP solvers. These proposals would work on a polyhedron whose vertices are also polyhedra, and the disjunctive and non-binary CSPs are solved by means of metaheuristic techniques. Also it is appropriate to generate algorithms to be configured dynamically depending on user inputs. This proposal will be composed of HSA as a complete algorithm, helped by heuristics like *POLYSA* and NFHH [13]. Thus, when the user insert the problem parameters, this proposal will study these parameters and will apply the most suitable algorithm.

References

1. Bacchus F., van Beek P.: On the conversion between non-binary and binary constraint satisfaction problems. In proceeding of AAAI-98, (1998) 311-318
2. Bessière C., Meseguer P., Freuder E.C., Larrosa J.: On Forward Checking for Non-binary Constraint Satisfaction. In Proc. Principles and Practice of Constraint Programming (CP-99), (1999) 88-102
3. Bessière C.: Non-Binary Constraints. In Proc. Principles and Practice of Constraint Programming (CP-99), (1999) 24-27
4. Dechter, R., Meiri, I., Pearl, J.: Temporal Constraint Network, Artificial Intelligence 49, (1991) 61-95
5. Gerevini A., Schubert L., Schaeffer S.: Temporal Reasoning in Time Graph I-II, SIGART Bull 4 (3) (1993) 21-25
6. Kautz H., Ladkin P.: Integrating metric and temporal qualitative temporal reasoning. In Proc. 9th National Conference Artificial Intelligence (AAAI-91), (1991) 241-246
7. Koubarakis M.: Dense time and temporal constraints with ≠. In Proc. 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92), (1992) 24-35
8. Larrosa J.: Algorithms and Heuristics for total and partial Constraint Satisfaction, Phd Dissertation, Barcelona, Spain, 1998
9. Meiri I.: Combining qualitative and quantitative constraints in temporal reasoning. Artificial Intelligence 87 (1996) 343-385
10. Rossi F., Petrie C., Dhar V.: On the equivalence of constraint satisfaction problems. In proceeding of European Conference of Artificial Intelligence, ECAI-90, (1990) 550-556
11. Salido M. A., Giret A., Barber F.: Constraint Satisfaction by means of Dynamic Polyhedra. In Operations Research Proceeding (OR2001), (2001)
12. Salido M.A., Barber F.: An Incremental and Non-binary CSP Solver: The Hyperpolyhedron Search Algorithm. In Proceedings of Seventh International Conference on Principles and Practice of Constraint Programming , (CP2001), 2001.
13. Salido M. A., Giret A., Barber F.: A combination of AI and OR methods for solving Non-binary Constraint Satisfaction problems. In Proceedings of Joint German/Austrian Conference on Artificial Intelligence (KI-2001) Workshop on New Results in Planning, Scheduling and Design (PUK2001). 2001