Realizing a Global Hyperpolyhedron Constraint via LP Techniques

Miguel A. Salido, Adriana Giret, Federico Barber

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

Abstract. Combination of AI/OR methods is gaining a great deal of attention because many combinatorial problems especially in planning and scheduling areas can be solved by means of combined AI/OR techniques. Many of these combinatorial problems can be expressed in a natural way as a Constraint Satisfaction Problem (CSP). It is well known that a non-binary CSP can be transformed into an equivalent binary one using some of the actual techniques. However, when the CSP is not discrete or the number of constraints is high, these techniques become impractical. In this paper, we propose an heuristic called "*N-face Hyperpolyhedron Heuristic*" as an incremental and non-binary CSP solver. This non-binary CSP solver carries out the search through OR techniques and maintains, in a hyperpolyhedron, those solutions that satisfy all metric non-binary constraints. Thus, we can manage more complex and expressive problems with high number of constraints and very large domains.

1 Introduction

Nowadays, many researchers are working on combined methods of AI/OR in order to manage problems that require both AI/OR techniques. In Artificial Intelligence environment many researchers are working on non-binary constraints satisfaction problems, mainly influenced by the growing number of real-life applications. 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. Problems of these kind can either be solved directly by means of non-binary CSPs by a search method or transformed into a binary one [6] and then solved by using binary CSP techniques. However, this transformation may not be practical in problems with some particular properties [1][3], for example when the number of constraints is high relative to the number of variables, when the constraints are not tight or when the CSP is non-discrete [2].
In this paper, we propose a propagation algorithm called "*N-face Hyperpolyhedron Heuristic*" (NFHH) that manages non-binary CSPs . In NFHH, the handling of the non-binary constraint (linear inequations) can be seen as one global hyperpolyhedron constraint [5]. Initially NFHH maintains only n new vertices in each

hyperpolyhedron face. Thus, the complexity of the NFHH is reduced to O(n^2), while the complete hyperpolyhedron algorithm (HSA) [7] has an exponential complexity.

This hyperpolyhedron is managed by means of OR techniques where the non-binary constraints are hyperplanes that are intersected to obtain the hyperpolyhedron vertices. However, AI traditional CSP techniques obtain the solution by searching systematically through the possible assignments of values to variables. The complexity of these AI techniques increase exponentially with the domain length, so it is necessary to use methods guided by heuristics. Combining AI/OR techniques we can obtain methods, like NFHH, that do not depend on the variable domains.

Thus, NFHH overcomes some of the weaknesses of other techniques. Moreover, it can manage constraints that can be inserted incrementally into the problem without needing to solve the whole problem again.

2 Preliminaries

Briefly, a constraint satisfaction problem (CSP) consists of:

- a set of *variables* $X = \{x_1, ..., x_n\}$;
- each variable $x_i \in X$ has got a finite set D_i of possible values (its *domain*);
- and a finite collection 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, in such a way that every constraint is satisfied.

The objective may be: get only one solution, with no preference as to which one; get some or all solutions; get an optimal, or a good solution by means of an objective function defined in terms of some variables.

2.1 Notation and definitions

In this section, we will summarize the notation that it is used in this paper.

Generic: The number of variables in a CSP will be noted by *n*. The domain of the variable x_i will be noted 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 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 noted by $D_i = [l_i, u_i]$, so the domain length of the variable x_i is $u_i - l_i$.

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. The constraints that NFHH manages, extend the framework of simple temporal problems studied originally by Dechter, Meiri and Pearl [4] to consider linear relations of the form:

$$\text{Inequation: } \sum_{i=1}^{n} p_i x_i \le b \tag{1}$$

where x_i are variables ranging over continuous or discrete intervals $x_i \in [l_i, u_i]$, *b* is a real constant, and $n \ge 1$.

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 will mean a CSP where some or all of the constraints have arity of more than 2. *NFHH* is a CSP solver that manage non-binary constraints.

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

$(2x_1 - 5x_2 + 3x_3 - 9x_4 \le 4)$, $(x_1 - 2x_2 - 4x_3 + x_4 = 4)$, $(3x_1 + 6x_3 - 2x_4 - 3x_5 \ge 4)$

The first constraint is managed directly by *NFHH*, the remaining ones are transformed in:
$(x_1 - 2x_2 - 4x_3 + x_4 = 4) \Rightarrow (x_1 - 2x_2 - 4x_3 + x_4 \le 4) \wedge (-x_1 + 2x_2 + 4x_3 - x_4 \le -4)$
$(3x_1 + 6x_3 - 2x_4 - 3x_5 \ge 4) \Rightarrow (-3x_1 - 6x_3 + 2x_4 + 3x \le -4)$

3 Specification of the N-Face Hyperpolyhedron Heuristic

In this paper we assume a non-binary CSP where *variables* x_i are bounded in continuous or discrete domains (for example: $x_i \in [l_i, u_i]$) and a collection of non-binary *constraints* of the form (1).

The specification of the *N-face Hyperpolyhedron Heuristic* (NFHH) is presented in (Fig. 1) (Fig. 2). Initially, NFHH has a static behaviour such as a classic CSP solver. The hyperpolyhedron vertices are generated by means of the partial Cartesian Product of the variable domains bounds ($D_1 \times D_2 \times \cdots \times D_n$) (step 1) such that, only *n* new vertices are assigned to each hyperpolyhedron face. Thus, the algorithm assigns to each face *n* vertices that have not ever been assigned in any adjacent face.

For each constraint, NFHH carries out the consistency check (step 2). If the constraint is not consistent, NFHH returns '*not consistent problem*'; else, NFHH determines if the constraint is not redundant, updating the hyperpolyhedron (step 3) by means of OR techniques like linear programming.

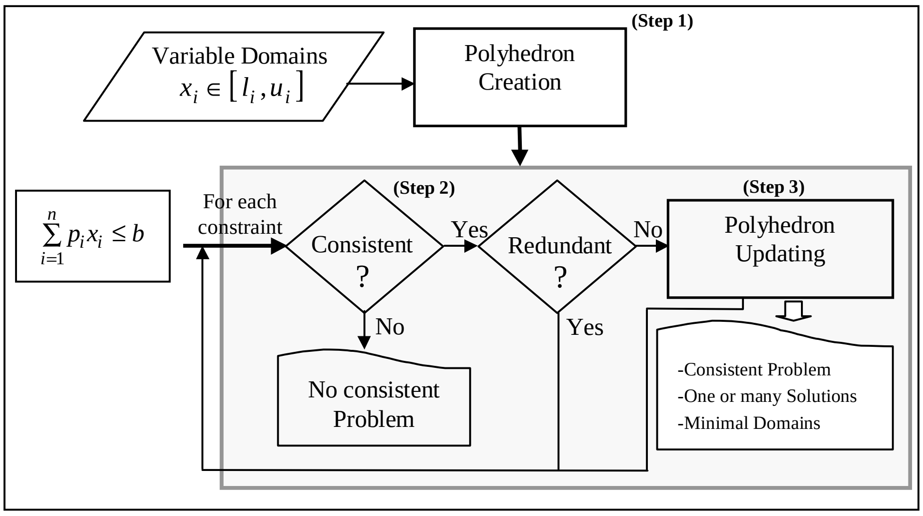


Fig. 1. General Scheme of the N-face Hyperpolyhedron Heuristic.

Finally, the objective of NFHH is to obtain some important results such as:

- the problem consistency, (i.e.) if the problems is consistent or not;
- one or many problem solutions, because OFHH maintains the convex set of solutions that satisfy all the constraints;
- the new variable domains;
- the solution that minimises or maximises some objective function.

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

The objective of selecting *n* vertices in each face is because there is a set of problems in which NFHH has a better behaviour than the other heuristics like OFHH [8] and POLYSA [9]. OFHH only assign one vertex in each hyperpolyhedron face and POLYSA assign n^2 vertices. OFHH has a linear complexity and is able to solve problems with many variables and very few constraints. However POLYSA is a cubic algorithm that manages problems with few variables and constraints. NFHH is an intermediate algorithm. Its complexity is quadratic and it is able to manage problem with many variables and more constraints than OFHH, with a CPU time lower than POLYSA.

```
NFHH (Dimension, Domains, Constraints)
{
    Step 1  ListV ← Hyperpolyhedron_creation (Dimension, Domains);
    $L_{yes} \leftarrow \phi$; $L_{no} \leftarrow \phi$;
    Step 2 For each $C_i \in$ Constraints do:
    {
      $\forall v_i \in ListV$ do:
         {
         If Satisfy($C_i$, $v_i$) ← true then:
                  $L_{yes} \leftarrow L_{yes} \cup \{v_i\}$ ;          // $C_i$ is consistent with the system
         else $L_{no} \leftarrow L_{no} \cup \{v_i\}$ ;
         }
      If $L_{yes} = \phi \Rightarrow STOP$ ;                    // $C_i$ is not consistent with the system
      If $L_{no} = \phi \Rightarrow$ " $C_i$ is consistent and redundant ";
      else
      Step 3 $\forall \bar{v} \in L_{no}$ Update_hyperpolyhedron ($\bar{v}$ , $L_{yes}$ , $L_{no}$ )
    }
NFHH returns the consistency check, and some extreme solutions
}
Hyperpolyhedron_creation (Dimension, Domains)
{
For i=1 to 2*Dimension do:              // for each hyperpolyhedron face do:
{
         For j=1 to Dimension do:       // NFHH assign to each face n vertices
         {
            Make(NewVertex);                  // New empty vertex is generated
            NewVertex ← Assign a new and not repeat vertex of the current face
            ListV = ListV ∪ NewVertex
         }
Return ListV ;                  // the module returns a list with 2*Dimensión$^2$ vertices.
}
Update_hyperpolyhedron ( $\bar{v}$ , $L_{yes}$ , $L_{no}$ )          // by means of LP techniques
{
         For each arc $\bar{a} = (\bar{v}, v)$ do:
         {
            NHH obtains the straight line $\bar{l}$ that unites both $\bar{v}$ and $v$ points.
            NHH intersects $\bar{l}$ with the hyperpolyhedron obtaining the new point $\bar{\bar{v}}$
            $L_{yes} \leftarrow L_{yes} \cup \bar{\bar{v}}$ ;
         }
return $L_{yes}$ ;
}
```

Fig. 2. N-face Hyperpolyhedron Heuristic.

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

Proof: NFHH is correct $\forall n \in \mathrm{N}$ because the resulting polyhedron is convex and it is a subset of the resulting convex hyperpolyhedron obtained by the complete algorithm HSA [7]. So, we can conclude that NFHH is correct $\forall n \in \mathrm{N}$.

Theorem 2*:* For $n < 7$, NFHH is complete.

Proof*:* Suppose by contradiction that NFHH is not complete for $n<7$.
The hyperpolyhedron generated by the Cartesian Product of the variable domains bounds, contains $2n$ faces and 2^n vertices. NFHH generates in each face *n unrepeat* vertices. Thus, NFHH generates $2n \cdot n = 2n^2$ vertices. If NFHH is not complete for $n<7$ then, there are some vertices that NFHH does not maintain, so $2n^2 < 2^n$ for $n<7$. This is a contradiction, because $2^n < 2n^2$ for $n<7$. So, NFHH is complete for $n<7$.

Example Let's see an example to visualize the hyperpolyhedron creation. The variables domains bounds can be projected to the interval [0,1] where 0 represents the lower bound and 1 the upper bound. Thus, the Cartesian Product of the new variable domain bounds generate a list of vertices labeled from 0 to 2^n that represent the binary number of each vertex.

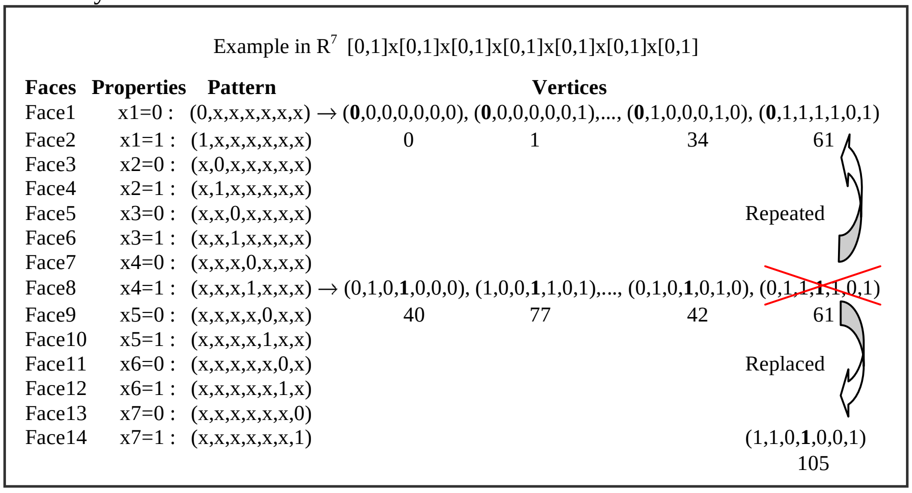

Example in R^7 [0,1]x[0,1]x[0,1]x[0,1]x[0,1]x[0,1]x[0,1]
Faces
Properties
Pattern
Vertices
Face1 x1=0 : (0,x,x,x,x,x,x) → (**0**,0,0,0,0,0,0), (**0**,0,0,0,0,0,1),..., (**0**,1,0,0,0,1,0), (**0**,1,1,1,1,0,1)
0 1 34 61
Face2 x1=1 : (1,x,x,x,x,x,x)
Face3 x2=0 : (x,0,x,x,x,x,x)
Face4 x2=1 : (x,1,x,x,x,x,x)
Face5 x3=0 : (x,x,0,x,x,x,x)
Face6 x3=1 : (x,x,1,x,x,x,x)
Face7 x4=0 : (x,x,x,0,x,x,x)
Face8 x4=1 : (x,x,x,1,x,x,x) → (0,1,0,**1**,0,0,0), (1,0,0,**1**,1,0,1),..., (0,1,0,**1**,0,1,0), ~~(0,1,1,**1**,1,0,1)~~
40 77 42 61
Face9 x5=0 : (x,x,x,x,0,x,x)
Face10 x5=1 : (x,x,x,x,1,x,x)
Face11 x6=0 : (x,x,x,x,x,0,x)
Face12 x6=1 : (x,x,x,x,x,1,x)
Face13 x7=0 : (x,x,x,x,x,x,0)
Face14 x7=1 : (x,x,x,x,x,x,1)
Repeated
Replaced
(1,1,0,**1**,0,0,1)
105

Fig. 3. Hyperpolyhedron generation in R^7.

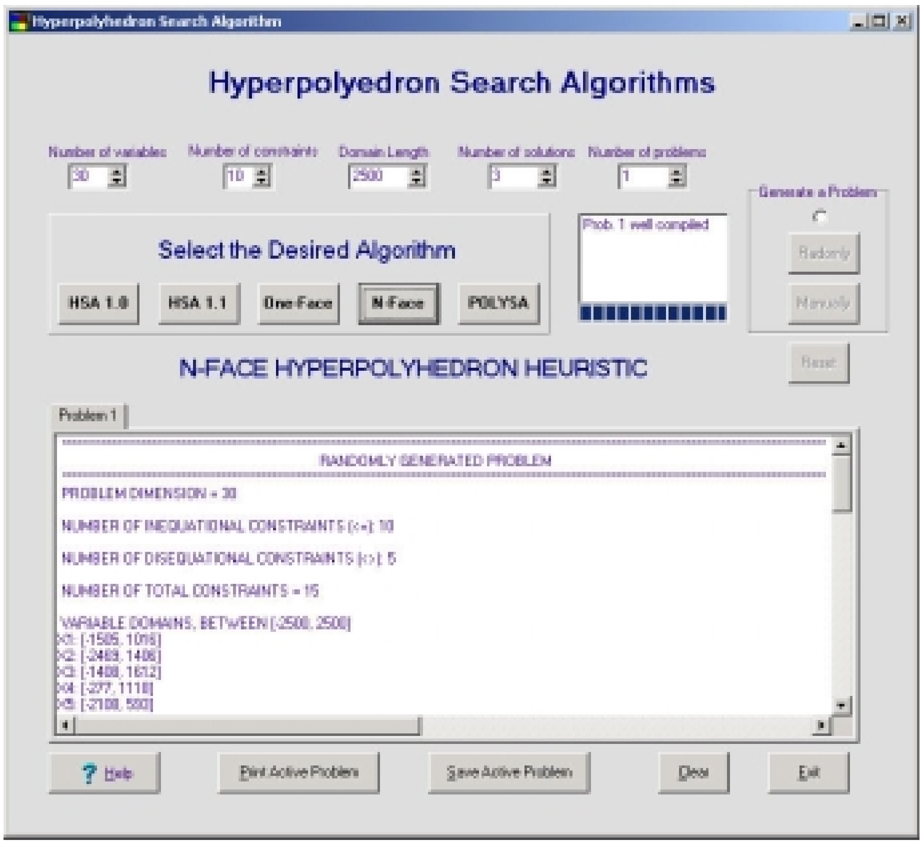
In Fig. 3 we can see the hyperpolyhedron generation. For each face random vertices are generated such that each one complies the pattern and has not been generated in a previous step.

3.1 Graphical Interface

NFHH allows the user to introduce the needed parameters to execute random or manually generated problems. In Fig 4 the graphical interface is presented. In the upper part of the screen it is showed the parameters that the user must configure: number of variables, number of constraints, domain length, number of desired solution if the problem is consistent, and finally number of problem if the user wants to generate several random problems. Then, when the parameters are fixed, the selected problems are randomly generated and solved by *NFHH*.

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

In the lower window of Fig. 4 the generated and solved problems are showed. This screen shows the selected parameters, the random or manually generated variable domains and the 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 showed.



Number of variables: 30
Number of constraints: 10
Domain Length: 2500
Number of solutions: 3
Number of problems: 1

Prob. 1 well compiled

```
RANDOMLY GENERATED PROBLEM

PROBLEM DIMENSION = 30

NUMBER OF INEQUATIONAL CONSTRAINTS (<=): 10

NUMBER OF DISEQUATIONAL CONSTRAINTS (<>): 5

NUMBER OF TOTAL CONSTRAINTS = 15

VARIABLE DOMAINS, BETWEEN [-2500, 2500]
X1: [-1505, 1016]
X2: [-2409, 1408]
X3: [-1408, 1612]
X4: [-277, 1118]
X5: [-2108, 500]
```

Fig. 4. Graphical Interface.

4 Analysis of the N-Face Hyperpolyhedron Heuristic

The NFHH spatial cost is determined by the number of vertices generated. Initially, the NFHH generates $2n^2$ vertices, where n is the number of problem variables. For each constraint (step 2), NFHH might generate n new vertices and eliminate only one. Thus, the number of hyperpolyhedron vertices is $2n^2+c(n-1)$ where c is the number of constraints. Therefore, the spatial cost is $O(n^2)$. However, other approaches, like HSA [7], increase their spatial cost to $O(2^n)$ because the algorithm generates all the hyperpolyhedron vertices and they becomes impractical in problems with many variables.

The temporal cost is divided in three steps: initialisation, consistency check and actualisation. The initialisation cost (step 1) is $O(n^2)$ because the algorithm only generates $2n^2$ vertices. For each constraint (step 2), the consistency check cost depends linearly on the number of hyperpolyhedron vertices, but not on the variable domains, so the temporal cost is $O(n^2)$. Finally, the actualisation cost (step 3) depends on the number of variables $O(n^2)$. Thus, the temporal cost is: $O(n^2) + c * \left(O(n^2) + O(n^2)\right) \Rightarrow O(n^2)$.

5 Evaluation of the N-Face Hyperpolyhedron Heuristic

In this section, we compare the performance of Forward-checking (FC), Real Full Look-ahead (RFLA)[1] and NFHH. In order to evaluate this performance, the computer used for the tests was a PIII-800 with 128 Mb. of RAM and Windows NT operating system.

The problems were randomly generated by modifying three parameters $<v,c,d>$, where v was the number of variables, c the number of constraints, and d the length of the variable domains. All the random constraints contain the whole set of variables. We generated three type of problems, fixing two parameters and varying the other parameter. We tested 100 problems for each value of the variable parameter, and we present the mean CPU time for these problems.

The graphics contain a horizontal asymptote in time=100 because we assigned a 100-second run time to those problems aborted because their run time exceeded 100 seconds. Also the problems that did not obtain the expected solution were assigned a 100-second run time in order to penalise its wrong behaviour.

In Fig. 5 the number of constraints and domain length were fixed $<v,2,2500>$, and the number of variables was increased from 10 to 26.

The graphic shows a global view of the behaviour of the algorithms. The mean CPU time in FC and RFLA is greater than NFHH. Also FC and RFLA were unable to solve more random problems than NFHH (see table 1.).

[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.

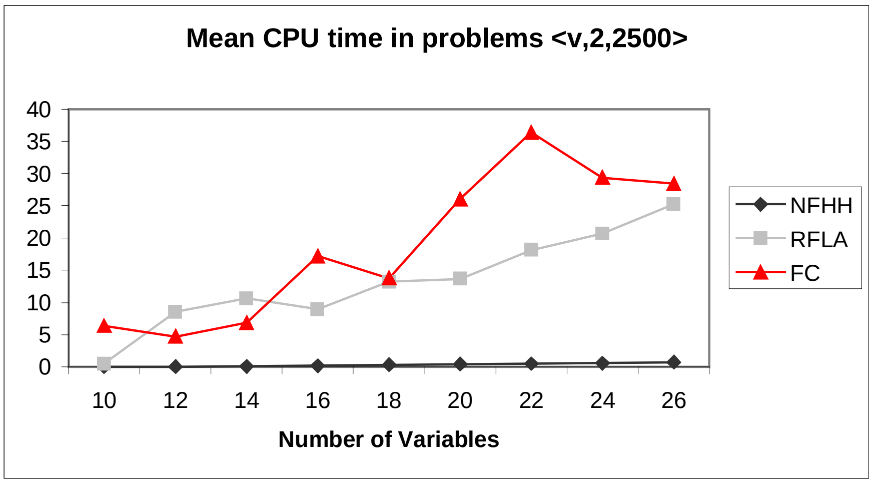


Fig. 5. Temporal Cost in problems <*v*,2,2500>.

In Fig. 6 the random domains and the number of variable were fixed <30,*c*,2500>, and the number of random constraints ranged from 2 to 6.

The graphic shows that FC and RFLA had similar performance and maintained an exponential temporal cost when the number of constraints increased while NFHH maintained its stable behaviours.

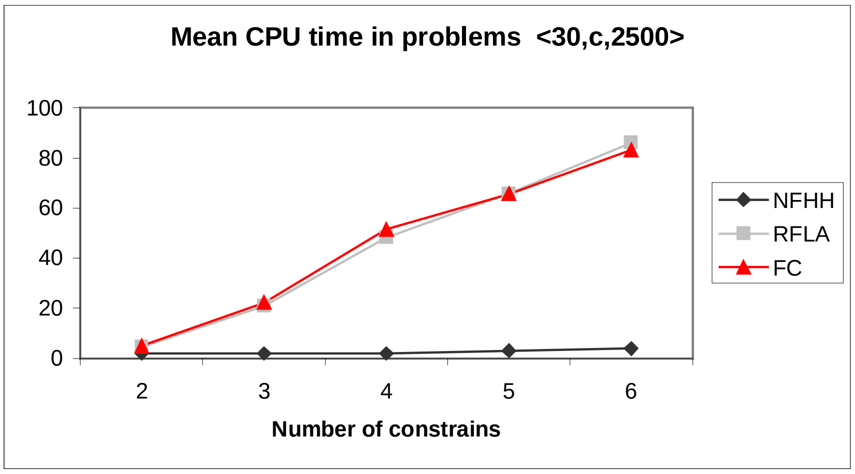


Fig.6. Temporal Cost in problems <30,c,2500>.

In Fig. 7 the number of variables and the number of random constraints were fixed <5,7,*d*>, and the domain length were increased from 500 to 8000.

The graphic summarises the performance of the algorithms. It can be observed that FC and RFLA maintained an exponential behaviour while NFHH did not increased it temporal cost because its complexity does not increased with the variables domain length.

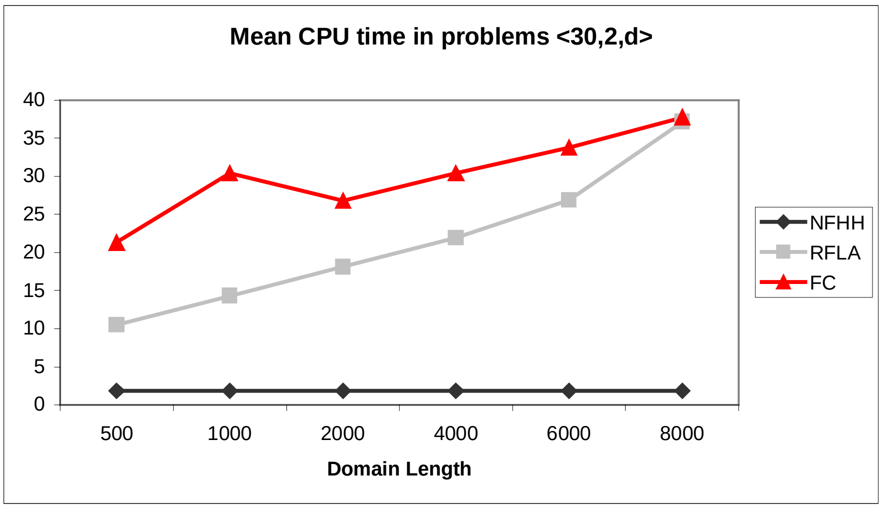


Fig. 7. Temporal Cost in problems <30,2,*d*>.

In table 1 the number of unsolved problems is presented. It can be observed that NFHH was unable to solve satisfactorily 3 of the whole 2100 problems while FC was unable to solve satisfactorily 281 and RFLA 249. For instance in problems <30,6,2500> FC and RFLA were unable to solve 80% and 76% respectively while NFHH only 2%.

Table 1. Number of unfinished or not well solved problems.

	V =	10	12	14	16	18	20	22	24	26
V	RFLA	0	4	5	4	6	6	8	9	11
C = 2	FC	3	2	3	8	6	12	17	13	12
D = 2500	NFHH	0	0	0	0	0	0	0	0	0

	C =	1	2	3	4	5	6
V = 30	RFLA	0	0	12	20	29	80
C	FC	0	0	7	22	29	76
D = 2500	NFHH	0	0	0	0	1	2

	D =	± 500	± 1000	± 2000	± 4000	±6000	±8000
V = 30	LA	3	5	7	9	14	17
C = 2	FC	7	12	10	12	14	16
D	NFHH	0	0	0	0	0	0

6 Conclusion and Future Works

In this paper, we have proposed an heuristic called NFHH as an incremental and non-binary TCSP solver. This proposal carries out the consistency study through a hyperpolyhedron that, by means of OR techniques, maintains in its vertices, those values that satisfy all metric temporal constraints. Thus, solutions to CSP are all vertices and all convex combination between any two vertices: (x_i and $x_j \Rightarrow \alpha x_i + (1-\alpha)x_j$, $\alpha \in [0,1]$).

We can conclude that combining AI/OR techniques several advantages can be obtained. NFHH is a CSP solver that manages non-binary constraints without needing to search systematically through the possible assignments of values to variables like AI traditional CSP techniques do.

In order to improve the behaviour of NFHH and due to the low temporal cost, we can run the not well solved problem several times in order to reduce the probability that NFHH fails. This technique is carried out varying the selected vertices that compose the hyperpolyhedron.

In future work, we will apply this technique to more complete heuristics in order to decrease the probability of unsolved problems. Currently, we are working on *mix-face hyperpolyhedron heuristic*s. This heuristic works on a hyperpolyhedron that mixes some heuristics like NFHH, OFHH [8] and POLYSA [9] in order to obtain a more suitable heuristic depending on the problem topology.

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 Proceeding on Principles and Practice of Constraint Programming (CP-99), (1999) 88-102
3. Bessiere, C.: Non-Binary Constraints. In Proceeding on. 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. Régin, J.C.: Minimization of the Number of Breaks in Sports Scheduling Problems Using Constraint Programming. In Proceedings of the DIMACS Workshop on Constraint Programming and Large Scale Discrete Optimization, (1998) 1-23.
6. 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
7. 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)
8. Salido, M.A., Giret, A., Barber, F.: A Non-binary Constraint Satisfaction Solver: The One-face Hyperpolyhedron Heuristic. In book: Research and Development in Intelligent Systems XVIII. (Ed. Springer Verlag). (2001)
9. Salido, M.A., Barber, F.: POLYSA: A Polynomial Algorithm for Non-binary Constraint Satisfaction Problems with <= and <>. Technical Report (DSIC-UPV), (2001)