Constraint Satisfaction by Means of Dynamic Polyhedra

Miguel A. Salido, Adriana Giret, Federico Barber

Universidad Politécnica de Valencia, 46071, Valencia, Spain

Abstract. *Nowadays, many real problems in the Artificial Intelligence environments, can be efficiently modelled as Constraint Satisfaction Problems (CSP's) and can be solved by means of Operational Research techniques. 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 an algorithm called "HSA≠" 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 researchers are working on combined methods of Artificial Intelligence and Operational Research (AI/OR) because many real problems can be modelled as constraint satisfaction problems (CSP's) and can be solved using linear programming techniques. These include problems from fields such as artificial intelligence, operational research, expert systems, databases, etc. Most of these problems can be naturally modelled using non-binary (or n-ary) constraints. In the constraint satisfaction literature, the need to address issues regarding non-binary constraints has only recently started to be widely recognised. Researchers have traditionally focused on binary constraints. The basic reasons were the simplicity of dealing with binary constraints compared to non-binary ones and the fact that any non-binary constraint satisfaction problem can be transformed into an equivalent binary one [5]. However, this transformation has several drawbacks:

- It should be mentioned that transforming a non-binary CSP into a binary one produces a significant increase in the problems size, so the transformation may not be practical [2][4].

- The translation process generates new variables, which may have very large domains, causing extra memory requirements for algorithms. So, in some problems, solving the binary formulation can be very inefficient [1].

In any case, this forced binarization generates unnatural formulations, which cause extra difficulties for constraint solver interfaces with human users [3].

In this paper, we propose an algorithm called "*The Hyperpolyhedron Search Algorithm*" (HSA≠) that handles non-binary CSPs by means of OR techniques where the non-binary constraints (inequalities) are hyperplanes that are intersected in order to obtain the polyhedron vertices. However, in Artificial Intelligent framework, traditional CSP techniques obtain the solution by searching systematically through the possible assignments of values to variables. These AI techniques increase their complexity exponentially with the domain length and the number of disequational constraints, so it is necessary to use methods which are guided by heuristics. By combining AI/OR techniques, we can obtain methods, like HSA≠, that do not depend on the variable domain length and on the number of disequational constraints.

Thus, HSA≠ 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 entire problem again.

2 Preliminaries

Briefly, the constraint satisfaction problem (CSP) that HSA≠ manages consists of:

- a set of variables $X = \{x_1, ..., x_n\}$;
- each variable $x_i \in X$ has a continuous 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, 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 or to find 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 some variables.

2.1 Notation and definitions

In this section, we 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 will be denoted by D_i. The constraints will be denoted by *c*, and all the constraint have the maximum arity *n*. We will always use this notation when analysing the behaviour of the 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 $u_i - l_i$. 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 αrb where r ∈ {<, ≤ , =, ≠, ≥, >}. A linear disequation over *X* is an expression of the form α≠b. A linear equality over *X* is an expression of the form α=b. The constraints that we are going to manage are linear relations and linear disequations of the form:

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

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

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 a conjunction of an inequality and an disequation. Thus, we can manage all possible combinations: {<, ≤ , =, ≠, ≥, >}.

3 Specification of the Hyperpolyhedron Search Algorithm

Initially, HSA≠ can be considered as a classic CSP solver, where there is a static set of constraints that the solutions must satisfy. The graph and pseudo-code specification of HSA≠ is presented in Figures 1 and 2, respectively. HSA≠ generates an initial polyhedron (step 1) with *2ⁿ* vertices created by means of the Cartesian Product of the variable domain bounds $\left([l_1,u_1]\times[l_2,u_2]\times\cdots\times[l_n,u_n]\right)$.

For each (≤) constraint, HSA≠ carries out the consistency check (step 2). If the (≤) constraint is not consistent, HSA≠ returns *not consistent problem*, else, HSA≠ determines if the (≤) constraint is not redundant, updating the polyhedron (step 3). This updating is carried out by Linear Programming techniques. The hyperplane generated by this (≤) constraint intersects with the polyhedron and thus, this hyperplane is the new face of the updated polyhedron. Finally, when all (≤) constraint are analysed and the problem is consistent with these (≤) constraints, HSA≠ checks the consistency with the (≠) constraints (step 4). The consistency study with the (≠) constraints does not update the polyhedron. This study only checks the consistency of the (≠) constraints with the polyhedron vertices.

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

Finally, HSA≠ can obtain important results such as: (1) the problem consistency; (2) one or all solutions; (3) the new variable domains; and (4) the vertex of the polyhedron that minimises or maximises an objective or multi-objective function.

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

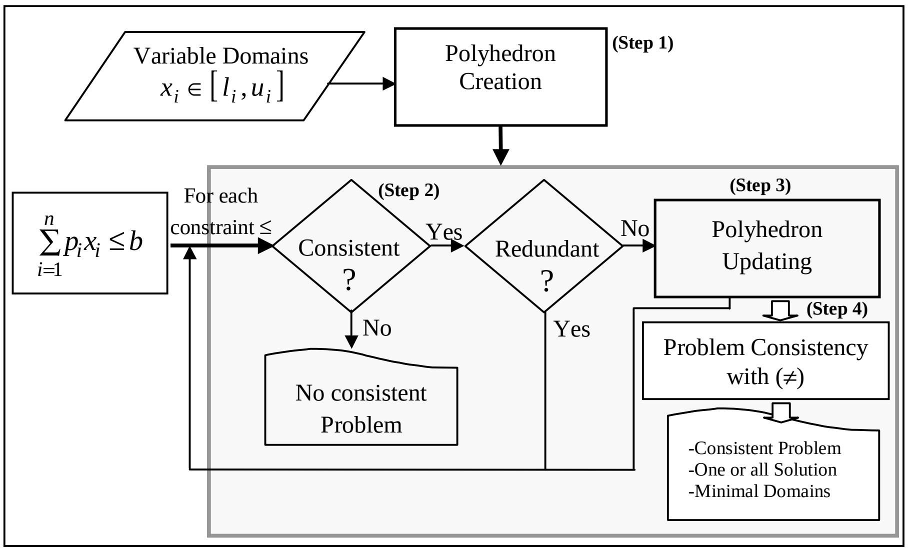

Fig. 1. General Scheme of the Hyperpolyhedron Search Algorithm.

```
HSA≠( Dimension, Domains, Constraints≤ , Constraints≠ , Solutions )
Step 1  ListV  ← Polyhedron_creation (Dimension, Domains, Constraints≤);
$L_{yes} \leftarrow \phi$; $L_{no} \leftarrow \phi$;
Step 2  For each $C_i \in$ Constraints≤  do:
  {
   $\forall v_i \in ListV$ do:
        {
        If Satisfy($C_i$, $v_i$) then:        // checks if the $v_i$ satisfies the constraint $C_i$
              $L_{yes} \leftarrow L_{yes} \cup \{v_i\}$;        // $C_i$ 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  $\forall v' \in L_{no}$ Polyhedron_Updation ($v'$, $L_{yes}$, $L_{no}$);
  }
Step 4  Satisfaction(Constraints≠, $L_{yes}$, Solutions);
return output; // HSA≠ returns the consistency check, and the extreme solutions
```

```
Polyhedron_Updation ($v'$, $L_{yes}$, $L_{no}$)
For each adjacent vertex $v$ of $v'$ do:
        HSA obtains the straight line $\bar{l}$ that unites both $v'$ and $v$ points.
        HSA intersects $\bar{l}$ with the polyhedron obtaining the new point $v''$
        $L_{yes} \leftarrow L_{yes} \cup v''$
return $L_{yes}$;
```

Fig. 2. Hyperpolyhedron Search Algorithm.

4 Analysis of Hyperpolyhedron Search Algorithm

The HSA≠ spatial cost is determined by the number of vertices generated. Initially, the HSA≠ generates 2^n vertices, where n is the number of problem variables. For each constraint (step 2), HSA≠ might generate n new vertices and eliminate only one. Thus, the number of polyhedron vertices is $2^n+c_{\leq}(n-1)$ where $c_{\leq}$ is the number of (≤) constraints. Therefore, the spatial cost is $O(c_{\leq}2^n)$.

The temporal cost is divided into three steps: initialisation, consistency check and actualisation. The initialisation cost (step 1) is $O(2^n)$ because the algorithm only generates 2^n vertices. For each (≤) constraint (step 2), the consistency check cost depends linearly on the number of polyhedron vertices, but not on the variable domains. Therefore the temporal cost is $O(2^n)$. Finally, the actualisation cost (step 3) depends on the number of variables $O(2^n)$. Also, the algorithm checks the consistency with each (≠) constraint in O(n).

Thus, the temporal cost is: $O(2^n)+c_{\leq}*\left(O(2^n)+O(2^n)\right)+c_{\neq}*O(n)\Rightarrow O(c_{\leq}2^n)$.

5 Evaluation of Hyperpolyhedron Search Algorithm

In this section, we compare the performance of HSA≠ with some of the more current CSP solvers. We have selected Forward-checking (FC) and Real Full Look-ahead (RFLA) because of they are the most appropriate techniques that are able to handle 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 inequational constraints, $c_{\neq}$ the number of disequational constraints and d the length of the variable domains. The problems were randomly generated by modifying the parameters. The constraints $c_{\leq}$ and $c_{\neq}$ are constraints type (1) and (2) respectively, with the coefficients $p_i\neq 0$.

Thus, each of the graphs shown, set three parameters and varied the other one in order to evaluate the algorithm performance when this last parameter increased. We tested 100 test cases for each problem and each value of the variable parameter, and we present the mean CPU time for each technique.

Four graphs are shown in Fig. 3 which correspond to the four significant parameters. The *unsolved problems* were assigned a 200-second run time. Thus, these graphs contain a horizontal asymptote in time=200.

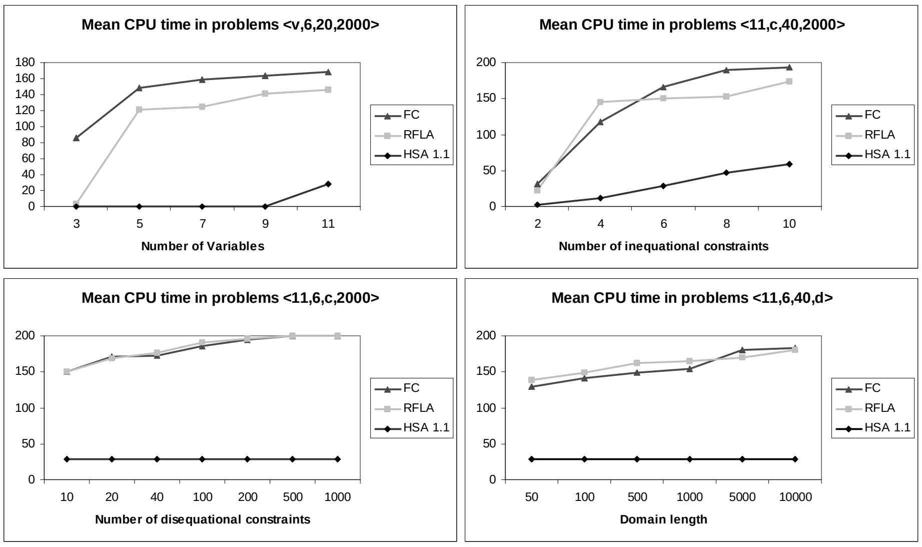

Fig. 3. Temporal Cost in problems with different parameters

Each graph in Fig. 3 shows that HSA≠ had better behaviour than FC and RFLA. The left upper graphic shows that HSA≠ increased it temporal cost later than the other approaches. The right upper graph summarises that our algorithm had a lower temporal cost than the others, when the number of inequational constraints increased. The lower graphs show that HSA≠ did not increase its temporal complexity when the number of disequational constraints and the domain length increased. However, FC and RFLA were unable to solve many of these problems.

C≤ = 6	V	3	5	7	9	11	
C≠ = 20	RFLA	0	52	54	63	65	
D=2000	FC	34	64	72	77	78	
V = 11	C≤	2	4	6	8	10	
C≠ = 40	RFLA	10	69	71	77	85	
D = 2000	FC	14	52	80	94	96	
V = 11	C ≠	10	20	40	100	200	500
C≤ = 6	RFLA	75	82	86	92	95	100
D = 2000	FC	75	85	86	92	96	100
V = 11	D	± 50	± 100	± 500	± 1000	± 5000	± 10000
C≠ = 6	RFLA	66	70	78	82	84	90
D = 2000	FC	62	70	74	76	88	90

Table 1. Number of unsolved problems

The number of unsolved problems is presented in table 1. HSA≠ was able to satisfactorily solve all the problems (2200 problems) while FC was unable to satisfactorily solve 1655 problems and RFLA 1546.

6 Conclusion and Future Works

In this paper, we have proposed an algorithm called HSA≠ as an incremental and non-binary CSP solver. This proposal carries out the consistency study through a polyhedron that is updated by means of LP techniques, and maintains (in its vertices) those values that satisfy all non-binary constraints. We are currently, 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 CSP's are solved by means of metaheuristics techniques. Also, it is appropriate to generate algorithms to be configured dynamically depending on the user's inputs. This proposal will be composed by HSA≠ as a complete algorithm, helped by heuristics like OFHH[6] and NFHH[7].

References

[1] Bacchus, F., van Beek, P. (1998) On the conversion between non-binary and binary constraint satisfaction problems. In proceeding of AAAI-98, 311-318

[2] Bessière, C., Meseguer, P., Freuder, E.C., Larrosa, J. (1999) On Forward Checking for Non-binary Constraint Satisfaction. In Proc. Principles and Practice of Constraint Programming (CP-99), 88-102

[3] Bessiere, C. (1999) Non-Binary Constraints. In Proc. Principles and Practice of Constraint Programming (CP-99), 24-27

[4] Larrosa J. (1998) Algorithms and Heuristics for total and partial Constraint Satisfaction, Phd Disertation, Barcelona, Spain,

[5] Rossi, F., Petrie, C., Dhar, V. (1990) On the equivalence of constraint satisfaction problems. In proceeding of European Conference of Artificial Intelligence, ECAI-90, 550-556

[6] Salido, M.A., Giret, A., Barber, F. (2001) A Non-binary Constraint Satisfaction Solver: The One-face Hyperpolyhedron Heuristic. In book: Research and Development in Intelligent Systems XVIII. (Ed. Springer Verlag).

[7] Salido, M.A., Giret, A., Barber, F. (2001) A combination of AI and OR methods for solving Non-binary Constraint Satisfaction problems. In Proceedings of Workshop on New Results in Planning, Scheduling and Design (PUK2001), 78-88