

# Exploiting the Constrainedness in Constraint Satisfaction Problems

Miguel A. Salido<sup>1</sup> and Federico Barber<sup>2</sup>

<sup>1</sup> Dpto. Ciencias de la Computación e Inteligencia Artificial, Universidad de Alicante  
Campus de San Vicente, Ap. de Correos: 99, E-03080, Alicante, Spain

`msalido@dsic.upv.es`

<sup>2</sup> Dpto. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia  
Camino de Vera s/n, 46071, Valencia, Spain

`fbarber@dsic.upv.es`

**Abstract.** Nowadays, many real problem in Artificial Intelligence can be modeled as constraint satisfaction problems (CSPs). A general rule in constraint satisfaction is to tackle the hardest part of a search problem first. In this paper, we introduce a parameter ( $\tau$ ) that measures the constrainedness of a search problem. This parameter represents the probability of the problem being feasible. A value of  $\tau = 0$  corresponds to an over-constrained problem and no states are expected to be solutions. A value of  $\tau = 1$  corresponds to an under-constrained problem which every state is a solution. This parameter can also be used in a heuristic to guide search. To achieve this parameter, a sample in finite population is carried out to compute the tightnesses of each constraint. We take advantage of this tightnesses to classify the constraints from the tightest constraint to the loosest constraint. This heuristic may accelerate the search due to inconsistencies can be found earlier and the number of constraint checks can significantly be reduced.

**keywords:** Constraint Satisfaction Problems, complexity, heuristic search.

## 1 Introduction

Many real problems in Artificial Intelligence (AI) as well as in other areas of computer science and engineering can be efficiently modeled as Constraint Satisfaction Problems (CSPs) and solved using constraint programming techniques. Some examples of such problems include: spatial and temporal planning, qualitative and symbolic reasoning, diagnosis, decision support, scheduling, hardware design and verification, real-time systems and robot planning.

These problems may be soluble or insoluble, they may be hard or easy. How to solve these problems have been the subject of intensive study in recent years.

Some works are focused on the constrainedness of search. Heuristics of making a choice that minimises the constrainedness can reduce search [3]. The constrainedness "knife-edge" that measures the constrainedness of a problem during search [10].

Most of the work are focused on general methods for solving CSPs. They include backtracking-based search algorithms. While the worst-case complexity of backtrack search is exponential, several heuristics to reduce its average-case complexity have been proposed in the literature [2]. For instance, some algorithms incorporate features such as variable ordering which have a substantially better performance than a simpler algorithm without this feature [5].

Many works have investigated various ways of improving the backtracking-based search algorithms. To avoid *thrashing* [6] in backtracking, *consistency* techniques, such as *arc-consistency* and *k-consistency*, have been developed by many researchers. Other ways of increasing the efficiency of backtracking include the use of *search order* for variables and values. Thus, some heuristics based on *variable ordering* and *value ordering* [7] have been developed, due to the additivity of the variables and values. However, constraints are also considered to be *additive*, that is, the order of imposition of constraints does not matter; all that matters is that the conjunction of constraints be satisfied [1]. In spite of the additivity of constraints, only some works have been done on constraint ordering heuristic mainly for arc-consistency algorithms [9, 4].

Here, we introduce a parameter that measures the "constrainedness" of the problem. This parameter called  $\tau$  represents the probability of the problem being feasible and identify the tightnesses of constraints. This parameter can also be applied in a heuristic to guide search. To achieve this parameter, we compute the tightnesses of each constraint. Using this tightnesses, we have developed a heuristic to accelerate the search. This heuristic performs a constraint ordering and can easily be applied to any backtracking-based search algorithm. It classifies the constraints by means of the tightnesses, so that the tightest constraints are studied first. This is based on the principle that, in goods ordering, domain values are removed as quickly as possible. This idea was first stated by Waltz [11] "*The base heuristic for speeding up the program is to eliminate as many possibilities as early as possible*" (p. 60).

An appropriate ordering is straightforward if the constrainedness is known in advance. However in the general case, an good classification is suitable to tackle the hardest part of the search problem first.

In the following section, we formally define a CSP and summarize some ordering heuristics and an example of soluble and insoluble problem. In section 3, we define a well-known definition of constrainedness of search problems. A new parameter to measure the constrainedness of a problem is developed in section 4. In section 5, we present our constraint ordering heuristic. Section 6 summarizes the conclusions and future work.

## 2 Definitions and Algorithms

In this section, we review some basic definitions as well as basic heuristics for CSPs.

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

- a set of variables  $V = \{v_1, v_2, \dots, v_n\}$

- each variable  $v_i \in V$  has a set  $D_{v_i}$  of possible values (its domain)
- a finite collection of constraints  $C = \{c_1, c_2, \dots, c_k\}$  restricting the values that the variables can simultaneously take.

A solution to a CSP is an assignment of values to all the variables so that all constraints are satisfied.

## 2.1 Constraint Ordering Algorithms

The experiments and analyses by several researchers have shown that the ordering in which variables and values are assigned during the search may have substantial impact on the complexity of the search space explored. In spite of the additivity of constraints, only some works have been done on constraint ordering. Heuristics of making a choice that minimises the constrainedness of the resulting subproblem can reduce search over standard heuristics [3].

Wallace and Freuder initiated a systematic study to identify factors that determine the efficiency of constraint propagation that achieve arc-consistency [9]. Gent et al. proposed a new constraint ordering heuristic in AC3, where the set of choices is composed by the arcs in the current set maintained by AC3 [4]. They considered the remaining subproblem to have the same set of variables as the original problem, but with only those arcs still remaining in the set. Walsh studied the constrainedness "knife-edge" in which he measured the constrainedness of a problem during search in several different domains [10]. He observed a constrainedness "knife-edge" in which critically constrained problems tend to remain critically constrained. This knife-edge is predicted by a theoretical lower-bound calculation.

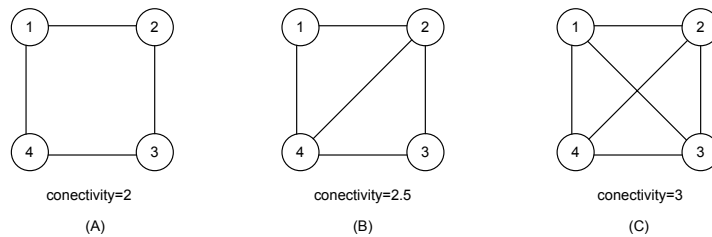
Many of these algorithms focus their approximate theories on just two factors: the size of the problems and the expected number of solutions. However, this last factor is not easy to estimate.

## 2.2 Solubility & Insolubility

Will a problem be soluble or insoluble? Will it be hard or easy to solve? How can we develop heuristics for new problem domains? All these questions have been studied by researchers in the last decades in a large number of problem domains. Consider the example presented in [3] of coloring a graph with a fixed number of colours so that neighboring nodes have different colours. If the nodes in the graph are loosely connected, then problems tend to be soluble and it is usually easy to obtain one of the many solutions. If nodes are highly connected, then problems tend to be insoluble and it is not easy to find a solution. At intermediate levels of connectivity, problems can be hard to solve since they are neither obviously soluble nor insoluble.

Figure 1 shows colouring graphs with four nodes using three colours: red, blue and green. Nodes connected by an edge must have different colours. The connectivity of a node is the number of edges connected to the node. The connectivity

of a graph is the average connectivity of its nodes. (A) is an under-constrained and soluble problem due to 18 states are solutions. (B) is a problem which is just soluble and 6 states are solutions but there is only an unique solution up to symmetry. (C) is an over-constrained and insoluble problem consisting of a clique of four nodes.



**Fig. 1.** Colouring Graph with four nodes using three colours.

We can use connectivity to develop a simple but effective heuristic for graph colouring that colours the most constrained nodes first. Consider colouring the nodes in Figure 1B without such a heuristic, using instead their numerical order. We might colour node 1 red, then node 2 green, and node 3 blue. We would then be unable to colour node 4 without giving it the same colour as one of its neighbors. Instead, suppose we seek to colour the most constrained nodes first. Both informally, the constrainedness of the graph is directly related to its connectivity. This then suggest the heuristic of colouring the nodes in decreasing order of their connectivity. As node 2 and 4 have the highest connectivity, they are coloured first. If we colour node 2 red and node 4 green, then nodes 1 and 3 can be coloured blue. Ordering the nodes by their connectivity focuses on the hardest part of the problem, leaving the less constrained and easier parts till last.

This example shows that some parts of the problem are hardest than others and a variable ordering may accelerate the search. However many real problems maintains thousand of variables and constraints and remains difficult to identify which variables and constraints are tightest. So a sample in finite populations may be useful to identify the constrainedness of the problem and perform a constraint ordering to take advantage of the problem topology. It may be useful to carry out domain filtering over the tightest constraints and even analyse the loosest constraints because many of them may be redundant and therefore removed.

### 3 Constrainedness $\kappa$

Given a new search problem, it is appropriate to identify parameters to measure the constrainedness of the problem and to develop heuristics for finding a solution more efficiently.

In [3], Gent et al. present a parameter that measures the constrainedness of an ensemble of combinatorial problems. They assume that each problem in an ensemble has a state space  $S$  with  $|S|$  elements and a number,  $Sol$  of these states are solutions. Any point in the state space can be represented by a  $N$ -bit binary vector where  $N = \log_2(|S|)$ . Let  $\langle Sol \rangle$  be the expected number of solutions averaged over the ensemble. They defined constrainedness,  $\kappa$ , of an ensemble by,

$$\kappa =_{def} 1 - \frac{\log_2(\langle Sol \rangle)}{N} \quad (1)$$

Gent et al. proposed a straightforward way to compute  $\langle Sol \rangle$  and therefore  $\kappa$  [3]. Consider constraint satisfaction problems, each variable  $v \in V$ , has a domain of values  $D_v$  of size  $d_v$ . Each constraint  $c_i \in C$  of arity  $a$  restricts a tuple of variables  $\langle v_1, \dots, v_a \rangle$ , and rules out some proportion  $\acute{p}_{c_i}$  of possible values from the cartesian product  $D_{v_1} \times \dots \times D_{v_a}$ . Without loss of generality we can define the tightnesses of a constraint as  $\acute{p}_c$  or its complementary  $1 - \acute{p}_c$ . Problems may have variables with many different domain sizes, and constraints of different arities and tightnesses.

The state space has size  $\prod_{v \in V} d_v$ . Each constraint  $c_i \in C$  rules out a proportion  $\acute{p}_{c_i}$  of these states, so the number of solution is

$$\langle Sol \rangle = \left( \prod_{v \in V} d_v \right) \times \left( \prod_{c_i \in C} (1 - \acute{p}_{c_i}) \right) \quad (2)$$

Substituting this into (1) gives

$$\kappa = \frac{-\sum_{c_i \in C} \log_2(1 - \acute{p}_{c_i})}{\sum_{v \in V} \log_2(d_v)} \quad (3)$$

$\kappa$  lies in the range  $[0, \infty)$ . A value of  $\kappa = 0$  corresponds to an under-constrained problem. A value of  $\kappa = \infty$  corresponds to an over-constrained problem.

However, this parameter defines the constrainedness of constraint satisfaction problems in general, but not of an individual problem.

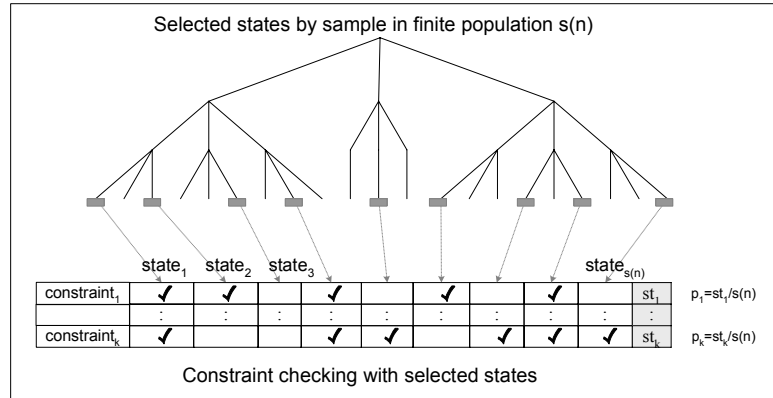
As we can observe, constrainedness in constraint satisfaction problem is closely related to probability. Unfortunately, it is difficult to compute this probability directly [3], mainly in a particular problem.

So, the main contribution of this papers focuses on computing this probability (or its complementary) for each constraint of the problem. These probabilities are computed by a sampling from finite populations, where there is a population, and a sample is chosen to represent this population. That is, the population is composed of the states lying within the convex hull of all initial states generated by means of the Cartesian product of variable domain bounds. The sample is composed by a set of random and well distributed states in order to represent the entire population. Each constraint is checked with the states of the sample and this gives us the probability  $p_{c_i}$  that constraint  $c_i$  satisfies the problem. Thus, we assume  $p_{c_i}$  the tightnesses of the constraint ( $p_{c_i} \equiv 1 - \acute{p}_{c_i}$ ). In this way, we can obtain the parameter  $\kappa$  or the parameter  $\tau$  developed in the following section.

## 4 Computing the constrainedness $\tau$

In this section, we introduce a parameter called  $\tau$  that measures the constrainedness of the problem. This parameter represents the probability of the problem being feasible. This parameter lies in the range  $[0, 1]$ . A value of  $\tau = 0$  corresponds to an over-constrained and no states are expected to be a solution ( $\langle Sol \rangle = 0$ ). A value of  $\tau = 1$  corresponds to an under-constrained and every state is expected to be a solution ( $\langle Sol \rangle = \prod_{v \in V} d_v$ ). This parameter can also be used in a heuristic to guide search. To this end, we take advantage of the tightnesses of each constraint to classifying them from the tightest constraint to the loosest constraint. Thus, a search algorithm can tackle the hardest part of the problem first.

To compute  $\tau$  a sample from a finite population is performed, where there is a population (states), and a sample is chosen to represent this population. The sample is composed by  $s(n)$  random and well distributed states where  $s$  is a polynomial function.



**Fig. 2.** From non-ordered constraint to ordered constraint

As in statistic, the user selects the size of the sample  $s(n)$ . We study how many states  $st_i : st_i \leq s(n)$  satisfy each constraint  $c_i$  (see Figure 2). Thus, each constraint  $c_i$  is labeled with  $p_{c_i} : c_i(p_{c_i})$ , where  $p_{c_i} = st_i / s(n)$  represents the proportion of possible states, that is, the tightnesses of the constraint.

In this way, given the set of probabilities  $\{p_{c_1}, \dots, p_{c_k}\}$ , the number of solutions can be computed as:

$$\langle Sol \rangle := \left( \prod_{v \in V} d_v \right) \times \left( \prod_{c_i \in C} (p_{c_i}) \right) \quad (4)$$

This equation is equivalent to the obtained in [3]. However, our definition of constrainedness is given by the following equation:

$$\tau := \prod_{c_i \in C} (p_{c_i}) \quad (5)$$

$\tau$  is a parameter that measures the probability that a randomly selected state is a solution, that is, the probability this state satisfies the first constraint ( $p_{c_1}$ ) and the probability this state also satisfies the second ( $p_{c_2}$ ) and so forth, the probability this state satisfies the last constraint ( $p_{c_k}$ ). Thus, this parameter lies in the range  $[0, 1]$  that represent the constrainedness of the problem.

We present the pseudo-code of computing  $\tau$ .

### Computing the constrainedness $\tau$

Inputs: A set of  $n$  variables,  $v_1, \dots, v_n$ ;

For each  $v_i$ , a set  $D_i$  of possible values (the domain)

A set of constraints,  $c_1, \dots, c_k$ .

Outputs: The constrainedness  $\tau$ .

- 1.- From the entire number of states generated by the Cartesian product of the variable domain bounds, a well distributed sample with  $s(n)$  states is selected.
- 2.- With the selected sample of states  $s(n)$ , we compute how many states  $st_i : st_i \leq s(n)$  satisfy each constraint  $c_i, i = 1..k$ . Thus,  $c_i$  is labelled with  $p_{c_i} = st_i/s(n)$ .
- 3.-  $\tau := \prod_{c_i \in C} (p_{c_i})$

For example, let's see the random problem presented in Table 1. There are three variables, each variable lies in the range  $[1,10]$ , and there are three constraints  $c_1, c_2, c_3$ . The first constraint satisfies 944 states, the second constraint satisfies 960 states and the third constraint satisfies 30 constraints. Thus the probability a random state satisfies the first constraint is  $p_{c_1} = \frac{944}{1000} = 0.944$ . Similarly,  $p_{c_2} = 0.960$  and  $p_{c_3} = 0.03$ . So, the probability a random state satisfies all constraints is 0.027. However, only 29 states satisfy these three constraints, so the above probability that a random state satisfies all constraint is 0.029.

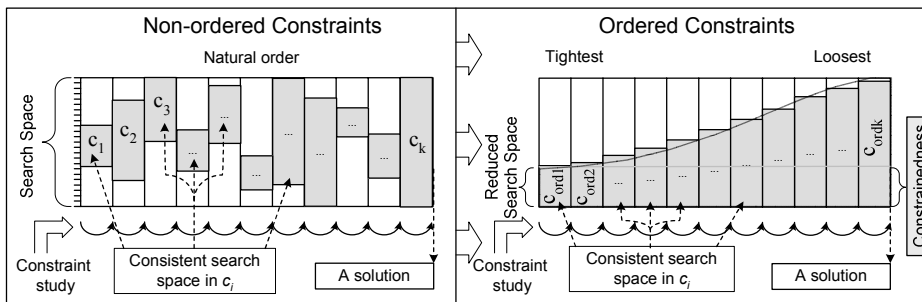
**Table 1.** random problem with 3 variables and 3 constraints

Z1,Z2,Z3: 1..10	Solutions	$p_{c_i}$	$\tau$
$c1 : -Z1 + Z2 + Z3 \leq 13$	944	0.944	<b>0.027</b>
$c2 : -Z1 - 2 * Z3 \leq -6$	960	0.96	
$c3 : Z2 + 3 * Z3 = 29$	30	0.03	
$c3 \& c2 \& c3$	29	<b>0.029</b>	dif=0.002

#### 4.1 A heuristic using $\tau$

To compute  $\tau$ , it is necessary to obtain the tightnesses of each constraint, represented by the following set of probabilities  $\{p_{c_1}, \dots, p_{c_k}\}$ . We take advantage of

this information to classifying the constraint so that a search algorithm can manage the hardest part of a problem first. Figure 3 shows the constraints in the natural order and classified by tightness. If the tightest constraints is very constrained ( $\tau \approx 0$ ), the problem will be over-constrained. However, if this tightest constraints is under-constrained ( $\tau \approx 1$ ) then, the problem will be under-constrained.



**Fig. 3.** From non-ordered constraints to ordered constraints: Constrainedness.

To achieve this objective, we classify the constraints in ascending order of the labels  $p_{c_i}$  so that the tightest constraints are classified first  $p_{c_{ord1}}, p_{c_{ord2}}, \dots, p_{c_{ordk}}$  (see Figure 3). Thus, a backtracking-based search algorithm can tackle the hardest part of a search problem first and inconsistencies can be found earlier and the number of constraint checks can significantly be reduced.

## 5 Evaluation of $\tau$ and the resultant heuristic

In this section, we evaluate our parameter  $\tau$  and our heuristic. To estimate the constrainedness of random problems we compare  $\tau$  with the actual constrainedness by obtaining all solutions of random problems. Furthermore, we study the performance of some well-known CSP solvers that incorporate our heuristic: Chronological Backtracking (BT), Generate&Test (GT), Forward Checking (FC) and Real Full Look Ahead (RFLA)<sup>3</sup>, because they are the most appropriate techniques for observing the number of constraint checks. This evaluation will be carried out on the classical  $n$ -queen problem.

### 5.1 Evaluating $\tau$

In our empirical evaluation, each random CSP was defined by the 3-tuple  $\langle n, c, d \rangle$ , where  $n$  was the number of variables,  $c$  the number of constraints and

<sup>3</sup> Backtracking, Generate&Test, Forward Checking and Real Full Look Ahead were obtained from CON'FLEX. It can be found in: <http://www-bia.inra.fr/T/conflex/Logiciels/adressesConflex.html>.

$d$  the domain size. The problems were randomly generated by modifying these parameters. We evaluated 100 test cases for each type of problem. We present the average actual constrainedness by obtaining all solutions, our estimator  $\tau$  choosing a sample of  $s(n) = 7n^2$  states, the number of possible states, the average number of possible solutions, the average number of estimate solutions using  $\tau$  and the error percentage.

Table 2 shows some types of random problems. For example in problems with 5 variables, each with 5 possible values and 5 constraints  $\langle 5, 5, 5 \rangle$ , the number of possible states is  $d^n = 5^5 = 3125$ , the average number of solutions is 125, so the actual constrainedness is 0.04. With a sample of  $7n^2 = 175$  states, we obtain an average number of 6.64 solutions. Thus, our parameter  $\tau = 0.038$  and the number of estimate solutions of the entire problem is 118.7. In this way, the error percentage is only 0.2%.

**Table 2.** Random instances  $\langle n, c, d \rangle$ ,  $n$ :variables,  $c$ :constraints and  $d$ :domain size

<i>Problems</i>	<i>actual con- strainedness</i>	<i>Parameter <math>\tau</math></i>	<i>Number of States</i>	<i>Number of Solutions</i>	<i>Number of Estimated Sol.</i>	<i>% Error</i>
$\langle 3, 5, 5 \rangle$	0.09	0.07	125	11.2	8.7	2%
$\langle 3, 5, 10 \rangle$	0.05	0.043	1000	50	43	0.7%
$\langle 3, 10, 5 \rangle$	0.024	0.013	125	3	1.6	1.12%
$\langle 5, 5, 5 \rangle$	0.04	0.038	3125	125	118.7	0.2%
$\langle 5, 10, 5 \rangle$	0.008	0.01	3125	25	31.2	0.19%
$\langle 5, 10, 10 \rangle$	0.0045	0.0034	100000	453	340	0.1%

## 5.2 Evaluating our heuristic

The  $n$ -queens problem is a classical search problem to analyse the behaviour of algorithms. Table 3 shows the amount of constraint check saving in the  $n$ -queens problem.

**Table 3.** Constraint check saving using GT , BT, FC and RFLA in the  $n$ -queens problem.

	<b>GT&amp;GT+CO</b>	<b>BT&amp;BT+CO</b>	<b>FC&amp;FC+CO</b>	<b>RFLA&amp;RFLA+CO</b>
<i>queens</i>	<i>Constraint Check Saving</i>	<i>Constraint Check Saving</i>	<i>Constraint Check Saving</i>	<i>Constraint Check Saving</i>
5	$2.1 \times 10^4$	$2.4 \times 10^2$	150	110
10	$4.1 \times 10^{11}$	$3.9 \times 10^7$	$1.4 \times 10^5$	$9.3 \times 10^4$
20	$1.9 \times 10^{26}$	$3.6 \times 10^{18}$	$9.6 \times 10^{14}$	$6.03 \times 10^{11}$
50	$2.4 \times 10^{70}$	$3.6 \times 10^{52}$	$3.1 \times 10^{44}$	$1.6 \times 10^{32}$
100	$2.1 \times 10^{143}$	$2.1 \times 10^{106}$	$4.5 \times 10^{93}$	$1.8 \times 10^{66}$
150	$5.2 \times 10^{219}$	$3.7 \times 10^{161}$	$6.8 \times 10^{142}$	$2.1 \times 10^{100}$
200	$9.4 \times 10^{295}$	$8.7 \times 10^{219}$	$9.9 \times 10^{198}$	$2.2 \times 10^{134}$

We incorporated our constraint ordering (CO) to well-known CSP solver: GT+CO, BT+CO, FC+CO and RFLA+CO. Here, the objective is to find all solutions. The results show that the amount of constraint check saving was significant in GT+CO and BT+CO and lower but significant in FC+CO and RFLA+CO. This is due to these techniques are more powerful than BT and GT.

## 6 Conclusion and future work

In this paper, we introduce a parameter ( $\tau$ ) that measures the "constrainedness" of a search problem.  $\tau$  represents the probability of the problem being feasible. A value of  $\tau = 0$  corresponds to an over-constrained problem.  $\tau = 1$  corresponds to an under-constrained problem. This parameter can also be used in a heuristic to guide search. To achieve this parameter, we compute the tightnesses of each constraint. We can take advantage of this tightnesses to classify the constraints from the tightest constraint to the loosest constraint. Using this ordering heuristic, the search can be accelerated due to inconsistencies can be found earlier and the number of constraint checks can significantly be reduced.

Furthermore, this heuristic technique is appropriate to solve problems as a distributed CSPs [8] in which agents are committed to solve their subproblems.

For future work, we are working on exploiting this constraint ordering to remove redundant constraints in large problems.

## References

1. R. Bartk, 'Constraint programming: In pursuit of the holy grail', in *Proceedings of WDS99 (invited lecture)*, Prague, June, (1999).
2. R. Dechter and J. Pearl, 'Network-based heuristics for constraint satisfaction problems', *Artificial Intelligence*, **34**, 1–38, (1988).
3. I.P. Gent, E. MacIntyre, P. Prosser, and T Walsh, 'The constrainedness of search', *In Proceedings of AAAI-96*, 246–252, (1996).
4. I.P. Gent, E. MacIntyre, P. Prosser, and T Walsh, 'The constrainedness of arc consistency', *Principles and Practice of Constraint Programming*, 327–340, (1997).
5. R. Haralick and Elliot G., 'Increasing tree efficiency for constraint satisfaction problems', *Artificial Intelligence*, **14**, 263–314, (1980).
6. V. Kumar, 'Algorithms for constraint satisfaction problems: a survey', *Artificial Intelligence Magazine*, **1**, 32–44, (1992).
7. N. Sadeh and M.S. Fox, 'Variable and value ordering heuristics for activity-based jobshop scheduling', *In proc. of Fourth International Conference on Expert Systems in Production and Operations Management*, 134–144, (1990).
8. M.A. Salido, A. Giret, and F. Barber, 'Distributing Constraints by Sampling in Non-Binary CSPs', *In IJCAI Workshop on Distributing Constraint Reasoning*, 79–87, (2003).
9. R. Wallace and E. Freuder, 'Ordering heuristics for arc consistency algorithms', *In Proc. of Ninth Canad. Conf. on A.I.*, 163–169, (1992).
10. T. Walsh, 'The constrainedness knife-edge', *In Proceedings of the 15th National Conference on AI (AAAI-98)*, 406–411, (1998).
11. D.L. Waltz, 'Understanding line drawings of scenes with shadows', *The Psychology of Computer Vision*, 19–91, (1975).