# Constrainedness and Redundancy by Constraint Ordering*

Miguel A. Salido[1] and Federico Barber[2]

[1] 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
[2] Dpto. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia
Camino de Vera s/n, 46071, Valencia, Spain
fbarber@dsic.upv.es

**Abstract.** In constraint satisfaction, a general rule 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 a 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 and every state is a solution. This parameter can also be used in heuristics to guide search. To achieve this parameter, a simple random or systematic sampling is carried out to compute the tightnesses of each constraint. New heuristics are developed to classify the constraints from the tightest constraint to the loosest constraint and to remove redundant constraints in constraint satisfaction problems. These heuristics may accelerate the search due to inconsistencies can be found earlier and the absence of such redundant constraints eliminate unnecessary checking and save storage space.

**keywords**: Constraint Satisfaction Problems, constrainedness, heuristics.

## 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.

* This work has been partially supported by the grant DPI2001-2094-C03-03 from the Spanish Government.

Some works are focused on the constrainedness of search. Heuristics of making a choice that minimises the constrainedness can reduce search [4].The constrainedness "knife-edge" measures the constrainedness of a problem during search [12].

Furthermore, some other works try to reduce a CSP by identifying and removing redundant constraints [10] since a constraint that allows all the possible value assignments of the variables on which it is defined, none of the constraints tuples has to be checked. Thus, the absence of such a constraint eliminates unnecessary checking and saves storage space [7]. However, identifying redundant constraint is hard, in general [10].

However, most of the work is 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 [3]. For instance, some algorithms incorporate features such as ordering heuristics. Thus, some heuristics based on *variable ordering* and *value ordering* [8] 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 be done on constraint ordering heuristic mainly for arc-consistency algorithms [11, 5].

Here, we introduce a parameter that measures the "constrainedness" of the problem. This parameter called $\tau$ represents the probability of a problem being feasible and identify the tightnesses of constraints. This parameter can also be applied in heuristics to guide search. To achieve this parameter, we compute the tightnesses of each constraint. Using this tightnesses, we have developed two heuristics to accelerate the search. These heuristics perform a constraint ordering and redundant constraints removal. They can easily be applied to any backtracking-based search algorithm.

The first one 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 [13] "*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, a good classification is suitable to tackle the hardest part of the search problem first.

The second heuristic is based on the idea of reducing a CSP into an "easy problem" by removing redundant constraints [10].

In the following section, we formally define a CSP and summarize some heuristics. In section 3, we present our parameter $\tau$ of constrainedness of search problems. Two heuristics using $\tau$ are developed in section 4. In section 5, we present an evaluation of $\tau$ and the proposed heuristics. Section 6 summarizes the conclusions and future work.

## 2 Definitions and Heuristics

In this section, we review some basic definitions as well as heuristics for constraint ordering, constrainedness and redundant constraints removing for CSPs.

### 2.1 Definitions

*Definition 1.* A constraint satisfaction problem (CSP) consists of:

- a set of variables $V = \{v_1, v_2, ..., v_n\}$
- each variable $v_i \in V$ has a set $D_{v_i}$ of possible values (its domain). We denote $d_{v_i}$ the length of domain $D_{v_i}$.
- a finite collection of constraints $C = \{c_1, c_2, ..., c_k\}$ restricting the values that the variables can simultaneously take.

*Definition 2.* A solution to a CSP is an assignment of values to all the variables so that all constraints are satisfied.

*Definition 3.* A redundant constraint is a constraint that can be removed without changing the solutions.

There is not a broadly accepted definition of constrainedness. We adopt the following definition of constrainedness:

*Definition 4.* The constrainedness of a problem is a predictor of computational cost to find a solution.

### 2.2 Heuristics

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 be done on constraint ordering.

Wallace and Freuder initiated a systematic study to identify factors that determine the efficiency of constraint propagation that achieve arc-consistency [11]. 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 [5]. 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.

On the other hand, many other problems may contain redundant constraints. Edward Tsang in [10] proposed the possibility of reducing a CSP to an "easy problem" by removing redundant constraints. However, redundancy is in general difficult to detect; but some redundant constraints are easier to identify than others. In [2], which focuses on binary CSPs, a number of concepts for helping to identify redundant binary constraints are introduced. For example, a constraint in a binary CSP can be removed if it is path-redundant (definition 3-19 [10]). The time complexity of path-redundant is $O(nd^3)$. However, since not

every problem can be reduced to an easier problem, one should judge the likelihood of succeeding in reducing the problem in order to justify the complexity of procedures such as path-redundant.

Heuristics of making a choice that minimises the constrainedness of the resulting subproblem can reduce search over standards heuristics [4]. Walsh studied the constrainedness "knife-edge" in which he measured the constrainedness of a problem during search in several different domains [12]. 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 which is difficult to obtain.

In [4], 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,

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

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

## 3 Constrainedness $\tau$

In this section, we introduce a parameter called $\tau$ that measures the constrainedness of the problem. This parameter represents the probability of a problem being feasible. This parameter lies in the range $[0, 1]$. A value of $\tau = 0$ corresponds to an over-constrained and no state is 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 and to removing some redundant constraints. Thus, a search algorithm can tackle the hardest part of the problem first with a lower number of constraints.

As we pointed out, a simple random or systematic sampling is performed to compute $\tau$, where there is a target population (states), and a sampled population is composed by $s(n)$ random and well distributed states where $s$ is a polynomial function.

As in statistic, the user selects the desired precision by 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 1). 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.

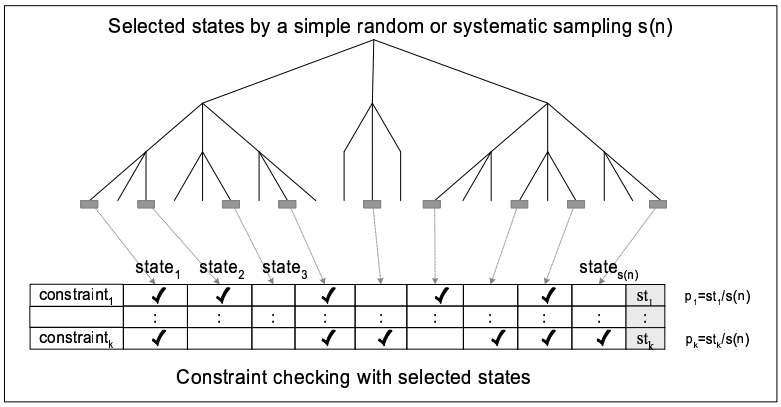


**Fig. 1.** From non-ordered constraint to ordered constraint

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

$$\langle Sol \rangle := (\prod_{v \in V} d_v) \times (\prod_{c_i \in C} (p_{c_i})) \tag{2}$$

This equation is equivalent to the obtained in [4]. However, our definition of constrainedness is given by the following equation:

$$\tau := \prod_{c_i \in C} (p_{c_i}) \tag{3}$$

$\tau$ 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 ($p_{c_1}$), the second constraint ($p_{c_2}$) 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, $v_1, ..., v_n$;
For each $v_i$, a set $D_i$ of possible values (the domain)
A set of constraints, $c_1, ..., c_k$.
Outputs: The constrainedness $\tau$.
1.- From the number of states generated by the Cartesian product of the variable domain bounds, a random and 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.- $\tau := \prod_{c_i \in C} (p_{c_i})$

## 4 Some heuristics using $\tau$

To compute $\tau$, it is necessary to obtain the tightnesses of each constraint, represented by the following set of probabilities $\{p_{c_1}, ..., p_{c_k}\}$. We can take advantage of this information to develops some heuristics to guide search or to improve efficiency.

The first heuristic is committed to classify the constraints so that a search algorithm can manage the hardest part of a problem first. Figure 2 shows the constraints in the natural order and classified by tightnesses. If the tightest constraints are very constrained ($\tau \approx 0$), the problem may be over-constrained. However, if these tightest constraints are under-constrained ($\tau \approx 1$) then, the problem will be under-constrained.

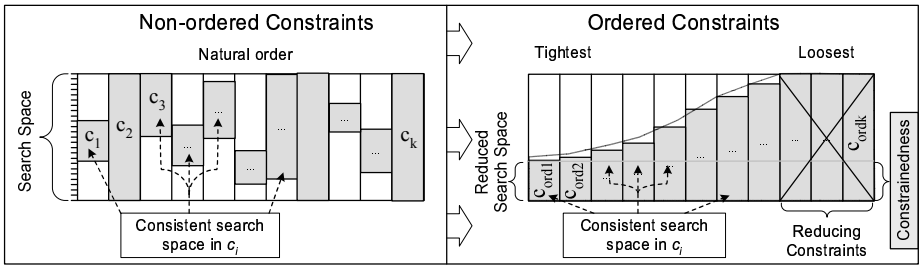


**Fig. 2.** From non-ordered constraints to ordered constraints: Constrainedness.

The second heuristic is focused on constraint removing in random CSPs. The loosest constraints are analyzed and the redundant ones are removed of the problem.

Let's see these two heuristics.

### 4.1 Constraint ordering heuristic

This easy heuristic takes advantage of $\tau$ making use of the probabilities of constraints $p_{c_1}, p_{c_2}, ..., p_{c_k}$. This heuristic classifies the constraints in ascending order of the labels $p_{c_i}$ so that the tightest constraints are classified first $p_{c_{ord1}}, p_{c_{ord2}}, ..., p_{c_{ordk}}$ (see Figure 2).

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.

### 4.2 Redundant constraint removing heuristic

In many random problems, some constraints may be redundant. A redundant constraint is a constraint that can be removed without changing the solutions. Some redundant constraint may be easier to identify than others [10].

In global constraints, to improve consistency a symbolic reasoning based on rewriting and redundant constraint introduction may help CSP solvers to find the result more directly [6].

However, in general, redundant constraints do not always save search effort [7]. For example, if a constraint allows all the possible value assignments of the variables on which it is defined, none of the constraints tuples has to be checked. The absence of such a constraint, in fact, eliminates unnecessary checking and saves storage space [7].

We will focus on this line, where our main goal is to identify redundant constraints in order to significantly reduce the number of constraint checks. Anyway, in case of global constraint, this heuristic may help to identify the set of redundant constraints.

We can identify two different types of redundant constraints:

- Constraints that are made redundant by any other single constraints (e.g., $x + y < 10$ is made redundant by $x + y < 5$).
- Constraints that due to their topology satisfy all possible assignments (e.g., $x + y < 5$ with domains $x : \{1, 2\}$ and $y : \{1, 2\}$).

The first type of constraints is not directly related to $\tau$ since the tightnesses $p_{c_i}$ of each constraint $c_i$ is not so relevant to identify redundancy.

The second type of constraints may be identified by the tightnesses of constraints. Constraint with $p_c = 1$ may be redundant due to all the random selected states in the sample satisfy this constraint. The coincidence can be given that all the selected states are consistent and however not to be a redundant constraint. So, we identify some types of constraints that can be removed if $p_c = 1$ and they satisfy a simple formula.

The main type of constraints is arithmetic constraints of the form:

$$\sum_{i=1}^{n} \alpha_i v_i \leq \gamma \tag{4}$$

Each constraint $c_i$ in the form (4) with $p_{c_i} = 1$ can be eliminated if:

$$\sum_{i=1}^{n} \alpha_i \beta_i \leq \gamma : \begin{cases} \beta_i = D_i^+ & \alpha_i > 0 \\ \beta_i = 0 & \alpha_i = 0 \\ \beta_i = D_i^- & \alpha_i < 0 \end{cases} \tag{5}$$

where $D^-$ and $D^+$ correspond to the lower and upper variable domain bound.

In this way, all constraints with $p_c = 1$ satisfying the above formula can be removed. The absence of such constraints eliminate unnecessary checking and save storage space [7].

## 5 Evaluation of $\tau$ and heuristics

In this section, we evaluate our parameter $\tau$ and both heuristics. To estimate the constrainedness of random problems we compare $\tau$ with the actual con-

strainedness by obtaining all solutions of random problems. To evaluate the constraint ordering heuristic we incorporated our heuristic to well-known CSP solvers: Backtracking (BT), Generate&Test (GT), Forward Checking (FC) and Real Full Look Ahead (RFLA)[3], because they are the most appropriate techniques for observing the number of constraint checks. Finally, we evaluated the redundant constraint removing heuristic over random problem using BT.

### 5.1 Evaluating $\tau$

In our empirical evaluation, each random CSP was defined by the 3-tuple: $< n, c, d >$, where $n$ was the number of variables, $c$ the number of constraints and $d$ 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 1 shows some types of random problems. For example in problems with 5 variables, each with 5 possible values and 5 constraints $< 5, 5, 5 >$, 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 1.** Random instances $< n, c, d >$, $n$:variables, $c$:constraints and $d$:domain size

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

### 5.2 Evaluating the constraint ordering heuristic

The $n$-queens problem is a classical search problem to analyse the behaviour of algorithms. Table 2 shows the amount of constraint check saving in the $n$-queens problem.

[3] BT, GT, FC and RFLA were obtained from CON'FLEX. It can be found in: http://www-bia.inra.fr/T/conflex/ Logiciels/adressesConflex.html.

**Table 2.** Constraint check saving using GT , BT, FC and RFLA in the $n$-queens problem.

| | **GT&GT+CO** | **BT&BT+CO** | **FC&FC+CO** | **RFLA&RFLA+CO** |
|---|---|---|---|---|
| *queens* | *Constraint Check Saving* | *Constraint Check Saving* | *Constraint Check Saving* | *Constraint Check Saving* |
| 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.

### 5.3 Evaluating the redundant constraint removing heuristic

We evaluated this heuristic over random problems as presented in section 5.1. In this case, all constraints are global constraints, that is, all constraints have arity $n$. Table 3 shows for each type of constraints, the average number of redundant constraints ($red_c$), the amount of constraint checks for the entire problem, the amount of constraint checks for the filtered problem (without redundant constraints) and the amount of constraint checks saving using backtracking (BT). It can be observed that the amount of constraint checks for the entire problem was $d^n c$. The constraint checks for the filtered problem was $d^n(c - red_c)$. So, the constraint check saving was $d^n(red_c)$, that corresponds a saving of 35%. These formulas may be standardized for backtracking. If the average number of redundant constraints is known, the constraint check saving can be obtained by $d^n(red_c)$,

**Table 3.** The redundant constraint removing heuristic in random instances $< n, c, d >$.

| *Problems* | *Redundant constraints* | *Constraint checks (entire problem)* | *Constraint checks (filtered problem)* | *Constraint checks saving* |
|---|---|---|---|---|
| $< 5, 5, 5 >$ | 1.75 | 15625 | 10325 | 5300 |
| $< 5, 10, 5 >$ | 3.5 | 31250 | 20312 | 10938 |
| $< 5, 20, 5 >$ | 7 | 62500 | 40625 | 21875 |
| $< 5, 5, 10 >$ | 1.75 | $5 \times 10^5$ | $3.3 \times 10^5$ | $1.7 \times 10^5$ |
| $< 5, 10, 10 >$ | 3.5 | $1 \times 10^6$ | $6.5 \times 10^5$ | $3.5 \times 10^5$ |
| $< 5, 20, 10 >$ | 7 | $2 \times 10^6$ | $1.3 \times 10^6$ | $7 \times 10^5$ |

## 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 a 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 and to remove redundant constraints. Using the constraint ordering heuristic and the redundant constraint removing heuristic, the search can be accelerated due to inconsistencies can be found earlier and the number of constraint checks can significantly be reduced.

Furthermore, these heuristic techniques are appropriate to solve problems as a distributed CSPs [9] in which agents are committed to solve their subproblems.

For future work, we are working on integrating constraint ordering with variable ordering in centralized and distributed CSPs.

## References


1. R. Bartk, 'Constraint programming: In pursuit of the holy grail', *in Proceedings of WDS99 (invited lecture), Prague, June*, (1999).
2. A. Dechter and R. Dechter, 'Removing redundancies in constraint networks', *Proceeding of National Conference on Artificial Intelligence (AAAI)*, 105–109, (1987).
3. R. Dechter and J. Pearl, 'Network-based heuristics for constraint satisfaction problems', *Artificial Intelligence*, **34**, 1–38, (1988).
4. I.P. Gent, E. MacIntyre, P. Prosser, and T Walsh, 'The constrainedness of search', *In Proceedings of AAAI-96*, 246–252, (1996).
5. I.P. Gent, E. MacIntyre, P. Prosser, and T Walsh, 'The constrainedness of arc consistency', *Principles and Practice of Constraint Programming*, 327–340, (1997).
6. A. Liret, P. Roy, and F. Pachet, 'Constraints satisfaction and symbolic reasoning for reactive control systems', *CP Workshop on Constraints in Control*, (1999).
7. M. Sabin and E.C. Freuder, 'Detecting and resolving inconsistency and redundancy in conditional constraint satisfaction problems', *Configuration Papers from the AAAI Workshop, WS-99-05*, (1999).
8. 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).
9. 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).
10. E. Tsang, *Foundation of Constraint Satisfaction*, Academic Press, London and San Diego, 1993.
11. R. Wallace and E. Freuder, 'Ordering heuristics for arc consistency algorithms', *In Proc. of Ninth Canad. Conf. on A.I.*, 163–169, (1992).
12. T. Walsh, 'The constrainedness knife-edge', *In Proceedings of the 15th National Conference on AI (AAAI-98)*, 406–411, (1998).
13. D.L. Waltz, 'Understanding line drawings of scenes with shadows', *The Psychology of Computer Vision*, 19–91, (1975).