Genetic Algorithms for Dynamic Variable Ordering in Constraint Satisfaction Problems

H. Terashima-Marín, R. de la Calleja-Manzanedo, and M. Valenzuela-Rendón

Center for Intelligent Systems, Tecnológico de Monterrey
Ave. Eugenio Garza Sada 2501 Sur, Monterrey, Nuevo León 64849 Mexico
{terashima@itesm.mx,rlight_renecm@hotmail.com,valenzuela@itesm.mx}

Abstract A Constraint Satisfaction Problem (CSP) can be stated as follows: we are given a set of variables, a finite and discrete domain for each variable, and a set of constraints defined over the values that each variable can simultaneously take. The objective is to find a consistent assignment of values to variables in such a way that all constraints are satisfied. To do this, a deterministic algorithm can be used. However, the order in which the variables are considered in the search process has a direct impact in the efficiency of the algorithm. Various heuristics have been proposed to determine a convenient order, which are usually divided in two types: static and dynamic. This investigation in particular uses Genetic Algorithms as a heuristic to determine the dynamic variable ordering during the search. The GA is coupled with a conventional CSP solving method. Results show that the approach is efficient when tested with a wide range of randomly generated problems.

1 Introduction

A Constraint Satisfaction Problem [1] (CSP) is composed of a finite set of variables, a discrete and finite domain of values for each variable, and a set of constraints specifying the combinations of values that are acceptable. The aim is to find a consistent assignment of values to variables in such a way that all constraints are satisfied, or to show that a consistent assignment does not exist. Several deterministic methods exist in the literature to carry out this process [2,1], and solutions are found by searching systematically through the possible assignments to variables, usually guided by heuristics. Many investigations have shown that the order in which the variables are considered for instantiation in the search has a direct impact in its efficiency [3]. There is a wide range of practical problems that can be modeled as CSPs. Applications of the standard form of the problem have included theorem proving, graph coloring and timetabling, machine vision, and job-shop scheduling [1]. Various heuristics have been proposed in the literature to determine an appropriate variable ordering, which can be classified in two types: static and dynamic. The heuristics of Static Variable Ordering (SVO) generate an order before the search begins, and it is not changed thereafter. In the heuristics of Dynamic Variable Ordering (DVO), the order in which the next variable to be considered at any point depends on the current

Advances in Artificial Intelligence Theory
Research on Computing Science 16, 2005, pp. 35-44

state of the search. It has been observed that heuristics for DVO outperform those heuristics for SVO [4,3]. This article presents an investigation which uses a Genetic Algorithm (GA) [5] as a dynamic heuristic to determine the appropriate variable ordering during the search. The GA is used with a Forward Checking algorithm (FC) and in this scheme the FC algorithm calls the GA which decides the next variable (one or more) to be instantiated. Results of this approach are compared against three other heuristics that have been widely used in similar studies and have provided reasonable performance for a variety of problems.

The reminder of this article is organized as follows. The next section describes the proposed solution model. Section 3 presents the results obtained and their discussion when the model is tested over different instances of CSPs. Finally, in Section 4 the conclusions are included.

2 Methodology

This report presents a combination of aspects of Constraint Satisfaction and Evolutionary Computation. This association has been used before. For instance, recent work by Craenen et al. [6] presents a comparative study on the performance of different evolutionary algorithms for solving CSPs. Research by Eiben [7] also discusses a methodology and directions for developing hybrid approaches with both techniques. The work presented in this paper, however, establishes the connection in a different way by concentrating on the problem of dynamic variable ordering when solving constraint satisfaction problems.

We herein describe a model to define the instances of CSP problems used in this work; they are binary CSPs (problems in which the constraints involve only two variables) defined by a four-tuple $\langle n, d, p_1, p_2 \rangle$, where n is the number of variables, d is the domain associated with each variable (for this investigation it is the same for all variables), p_1 is the probability that there is a constraint between a pair of variables, and p_2 the probability that, given that there is a constraint between two variables, the pair of values is inconsistent. This means that p_1 and p_2 represent an approximation of constraints in the problem (*constraint density*) and a number of inconsistent pairs of values (*constraint tightness*), respectively. A problem of this kind will have $p_1 \frac{n(n-1)}{2}$ constraints, and $p_2 d^2$ over each constraint. The same model has been used in other similar studies [8,9,10].

As a basis for comparison, this work uses several variable ordering heuristics that have been previously studied. These algorithms are based on the principle of selecting the 'most constrained variable'; the heuristics attempt to fail as soon as possible when instantiating variables, what leads to reinstantiate the variables with other values, and so eliminate search subregions of considerable size. These heuristics are the following:

Brelaz. This heuristic was designed for solving graph coloring problems. For a partial coloring, the *saturation* degree of a vertex is the number of different colors used to color the adjacent vertices. For our problem, the heuristic selects first the variable with maximum saturation degree (the variable with fewer values in

its domain). Then, it breaks ties by selecting the variable with maximum degree (the degree for a variable is the number of adjacent uninstantiated variables).

Rho. This heuristic selects first the variable that maximizes equation $\rho = \prod_{c \in C - C_i}(1 - p_c)$. That is, the variable that minimizes $\prod_{c \in C_i}(1 - p_c)$, where C is the set of constraints in the problem, C_i is the set of incident constraints in the current variable V_i and if a constraint c in average limits a fraction p_c of possible assignments, a fraction $1 - p_c$ is allowed. Thus this heuristic selects first the variable with the most and/or tightest constraints. The idea behind it is that by selecting this variable, the remaining subproblem contains a larger number of solutions (solution density ρ). The heuristic, however, does not take into account the available domain of the variables.

Kappa. This heuristic selects the variable in such a way that the parameter κ is minimized, where κ is a measure over the subproblem left after extracting the variable V_i and is given by the following equation: $\kappa = \frac{-\sum_{c \in C} \log_2(1-p_c)}{\sum_{v \in V} \log_2(d_v)}$ where V is the set of variables in the problem, and d_v is the domain size of variable v. This heuristic depends on the proposal by Gent et al. [11] in which κ captures the notion of the constrainedness of an ensemble of problems. The problems with $\kappa \ll 1$ are likely to be under-constrained, and solvable, whereas if $\kappa \gg 1$, these problems are likely to be over-constrained and unsolvable. Similarly as the heuristic Rho, this heuristic intends to select a variable that will leave a subproblem with high probability of being solvable.

More formal description of each heuristic can be found in the work by Gent et al. [12].

2.1 Solution Approach

The method for solving the CSPs in this work is the Forward Checking (FC) algorithm. FC takes a variable from the uninstantiated ones, sets a value for it, and propagates constraints (it keeps consistency in the domains of the adjacent and remaining variables). If one of those variables finishes with an empty domain, then the algorithm chronologically backtracks (BT), otherwise it continues with the next variable. This algorithm was chosen because it provides updated information in relation to the unsolved subproblem in each iteration. This information is in fact used by the heuristics to determine the next variable to instantiate. In our hybrid approach, the FC algorithm invokes the GA which runs for a number of cycles and determines the variable (it may be one or more) to be instantiated. Figure 1 illustrates the implementation diagram.

The variable(s) to be instantiated by the FC algorithm are taken from the best individual in the last cycle of the GA, every time this is called. The selected variables are those placed to the left-most part of the chromosome (a permutation-based representation is used) where each gene represents the index of each uninstantiated variable. n is the number of variables in the chromosome.

When the FC algorithm starts solving a given instance of CSP, it calls the GA, which initializes the population (popsize is $15n$) with randomly generated

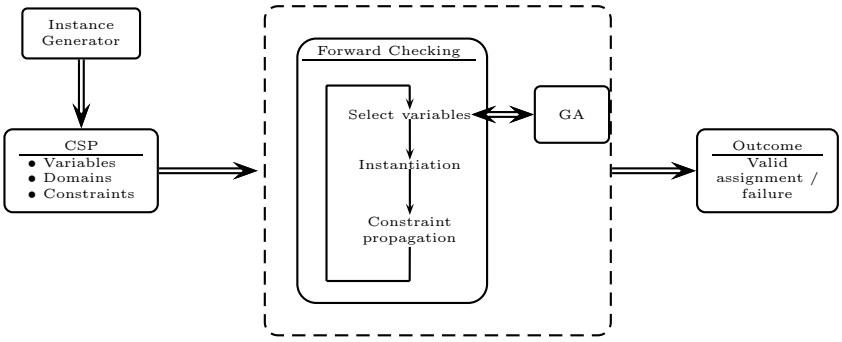


Figure 1. Flowchart of the proposed approach.

Table 1. Control parameters for the GA.

Parameter	Initial	Following
Population	$15n$	$10n$
Cycles	$12n$	$8n$
Replacement	60%	60%
Crossover Probability	90%	90%
Mutation Probability	10%	10%

chromosomes, runs for $12n$ cycles, and returns the selected variables for instantiation (*step*). FC is then used to assign values to *step* variables (the ones on the left), so there will be $n - step$ left to assign. For the subsequent invocations, the population in the GA is initialized based on the best chromosome of the previous call (we call it base chromosome and it is the one used to select the *step* variables). Now, the new population of size $10(n - step)$ is created, which is run for $8(n - step)$ cycles. The chromosome used to generate the new population is modified using random alterations. A copy of the base chromosome is also inserted in the population. The process continues until the complete CSP has been solved. The type of the proposed GA is steady state, with tournament selection, PMX crossover, and swap mutation. The GA was empirically tunned in its parameters and the final parameter set is presented in Table 1.

The objective function in the GA is given by the following expression:
$Ev = S_1 + nS_2$ where $S_1 = \sum_{i=1}^{step} \frac{T_i}{A_i(D_i)^2} {D_{max}}^2 (n-i)^2$ and
$S_2 = \sum_{j=step+1}^{n} D_j \left(\frac{j}{n}\right)^2$ being n the number of variables remaining to be instantiated, *step* the number of variables the GA returns to the FC algorithm to be instantiated, D_i the size of the current available domain for variable V_i, D_{max} is the largest domain associated to a variable, A_i is the number of adjacent variables to variable V_i and $T_i = \sum_{j=step+1}^{n} \frac{\text{conf}_{i,j}}{D_i D_j}$ where $\text{conf}_{i,j}$ is the number of pairs in conflict between the current available values for variables V_i and V_j.

The best individual is the one that maximizes the objective function above. This fitness function combines ideas from both the Brelaz and Kappa heuristics, specifically, with S_1 we are looking for those variables with small available domain and at the same time with constraints with high degree (with $\frac{T_i}{A_i}$), while

S_2 is used to emphasize that variables with small available domain should be selected first (shifted to the left side). From this expression, it can be observed that for smaller domains of the involved variables the value on S_1 would increase. It is also beneficial to have those variables to the left of the chromosome, and this is achieved by introducing the factor $(n-i)^2$. The value that makes a difference between two or more variables with the same minimal domain is T_i which is a sum of the tightness for each uninstantiated variable and adjacent to variable V_i. The quotient of T_i divided by A_i in S_1 would give us an idea of how 'hard' in average are the constraints linking V_i with the rest of the variables, considering only their available values. S_2 gives preference to select first those variables with smaller domain. This effect is achieved by maximizing the sum of the available domains of the remaining variables. Taking advantage on this, we also consider shifting variables with smaller available domains to the first positions in the chromosome. We give a weight to each position with $(j/n)^2$. It is also important to stress that when the FC algorithm is combined with heuristics Rho, Kappa, or Bz, just a single variable is returned for instantiation, that is, parameter *step* is only applies for the FC-GA combination.

3 Experiments and Results

This section presents the most important results obtained by the proposed approach. The experiments are divided following two different ways for generating the instances: in the first one, the generated problem instances have the parameter p_2 constant, that is, all constraints have the same number of inconsistent pairs; while in the second one, this parameter is randomly varied, leaving different number of inconsistent pairs between constrained variables. This way of generating the instances allows to observe the behavior of the various heuristics when the parameters used to define an instance are not uniform. For both cases, the performance of the algorithms is based on the number of consistency checks performed by the FC algorithm, with the aim to minimize it. The number of consistency checks is the most common way of comparing algorithms of this kind when solving constraint satisfaction problems, but there exist two other usual criteria such as the number of expanded nodes in the search tree and the number of backtracks. In the case of the GA, results report the average and best result over ten runs of the same instance.

Problem instances with uniform p_2
We present results for instances with 10 and 20 variables and domain size of 10. For both sizes, three different experiments are carried out, each one with a different value for parameter p_1 (constraint density).

First, we report results for instances with 10 variables. 40 random instances were generated for each value of p_2, the FC algorithm is executed with each heuristic and each instance, and then the average number of consistency checks is computed. That is the number plotted in the figures. For the GA case, each instance is run 10 times and then their average is used to obtain the average over the 40 instances, which is the one reported in the figures. p_2 is increased

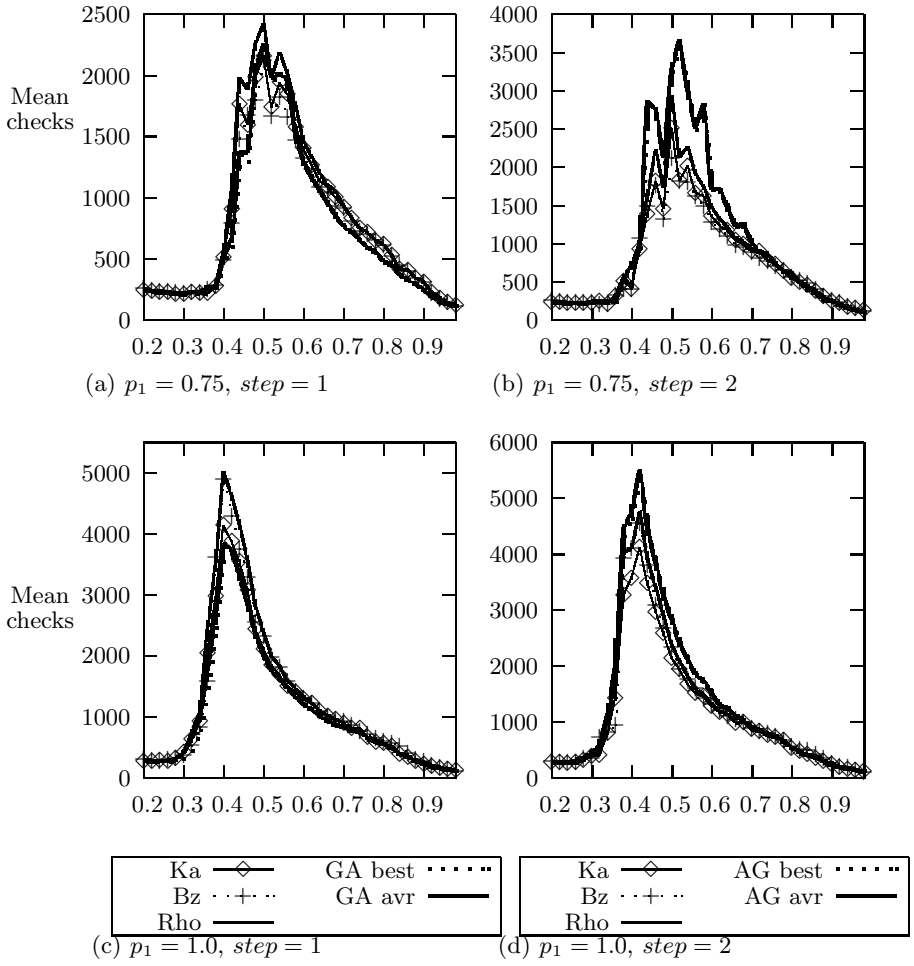


Figure 2. Result on problems $\langle 10, 10 \rangle$ with uniform p_2.

from 0.2 to 0.98 with steps of 0.02, trying to observe in detail the behavior of each heuristic in this range, and specifically over the phase transition (the region where the most difficult instances can be found). Figure 2 shows the first three series of experiments. In Figures 2 ((a) and (c)) the value for p_1 is 0.75 and 1, respectively.

The parameter *step* is set to 1. It can be observed that for all values of p_1, the GA approach shows slightly better results than the other single heuristics. It is also shown that there is an improvement from the approach when the instances have higher constraint density, for example, when p_1=1, it is clear that the FC-GA combination achieves better results (see Figure 2 (c)). Parameter *step* was intended to allow the FC-GA combination to select more than one variable to

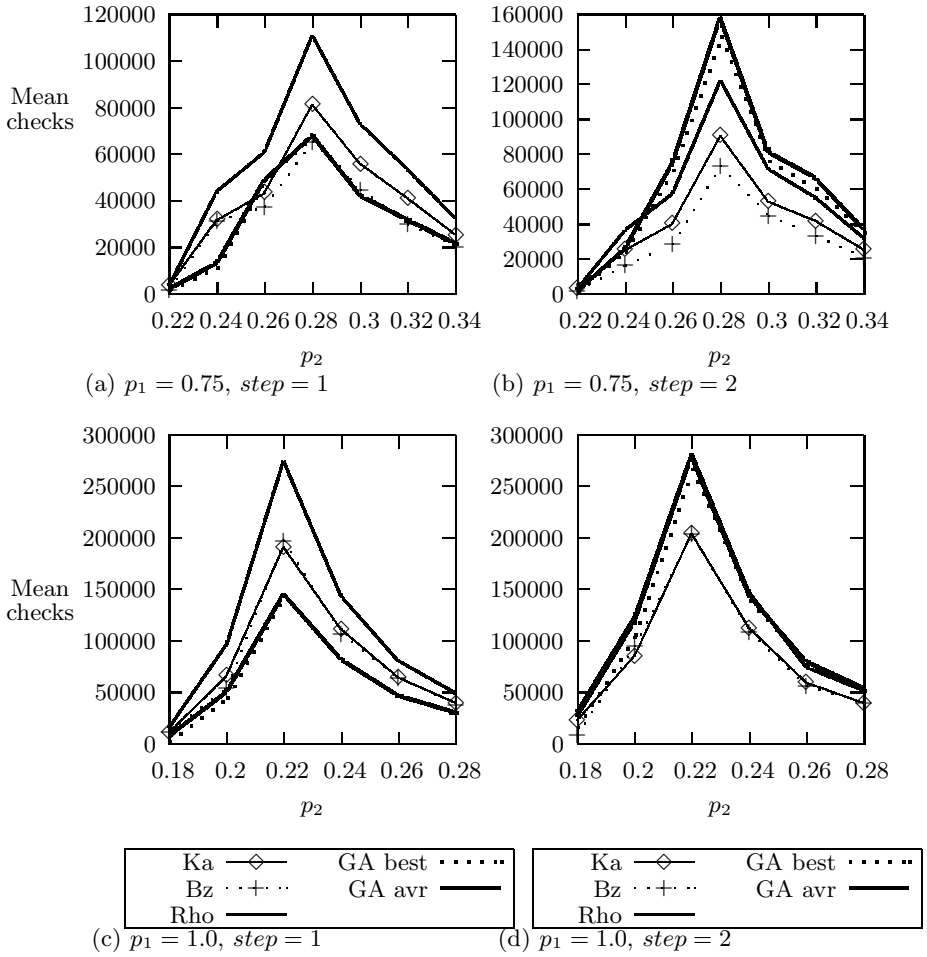


Figure 3. Results on problems $\langle 20, 10 \rangle$ with uniform p_2.

instantiate at each invocation. When we set $step = 2$ results can be observed in Figure 2 ((b) and (d)). We found for these cases and even when step had a higher value, single heuristics outperform our strategy.

The connection we deduced from these results is that indeed assigning a value to the most-left variable produces changes in the domains of the uninstantiated variables including that one selected by the parameter *step*. Consequently, those changes affect the domains of this variable, and so its selection is no longer the best one.

We now present results for instances with 20 variables. In this case, 20 different instances were randomly generated and tested with the FC algorithm and for each different value of p_2. The average number of consistency checks is reported in the figures. For the GA, the figures report the average of the average

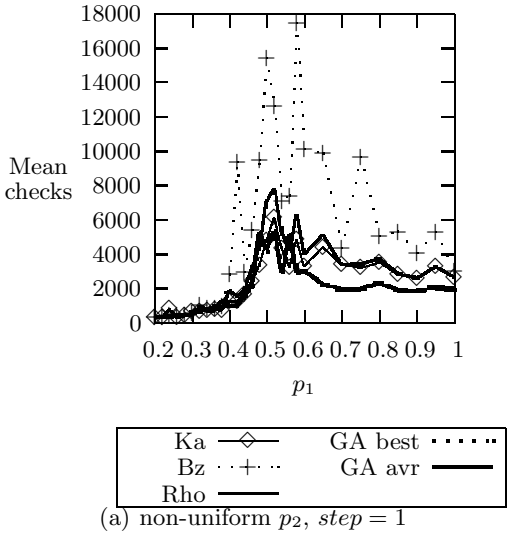
of ten runs for each instance, the same way as in the previous experiment. p_2 is increased by steps of 0.02. For these results, we concentrate in the range of values for p_2 between 0.22 and 0.34, where the phase transition appears. See Figure 3. It is clear in Figures (a) and (c) that the GA performs fewer number of constraint checks when *step*=1. Again, the GA shows better performance for highly constrained problems ($p_1 = 0.75$ and $p_1 = 1$). However, it is also true that as we increase the value in *step* to 2 (see Figure 3 (b) and (d)), the the advantage of the GA with respect to the other heuristics is less evident. In fact, we ran experiments for greater values of *step* (up to 5), but the best performance was found when *step*=1.

Problem instances with non-uniform p_2
In this set of experiments, the probability for inconsistent values between constrained pairs of variables (p_2) is not uniform. Because of this non-uniformity, a given variable may have more information, in addition to its available domain and degree, to be considered when selecting variables for instantiation. Heuristics Rho and Kappa, as well as our approach, exploit this situation. It is not the case with heuristic Bz, so that it is expected to produce different behavior in the results for the various heuristics.

For these experiments, instances have 20 variables, domain size of 10, and other additional particular features were considered. Specifically, for 15% of the constraints in an instance, parameter p_2 was set to 0.8, while for the remaining 85% of constraints, the same parameter has a value of 0.2. Now, what is interesting to observe is the behavior of the heuristics when the constraint density is varied (p_1). This parameter varies from 0.2 through 1 with steps of 0.02. For each value of p_1, 20 random instances were generated, each one was run 10 times, and the average number of consistency checks was computed.

Figure 4 (a) shows results when comparing all heuristics and the GA approach with *step*=1. As expected, the performance of heuristic Bz is very poor with respect to the other heuristics. The GA clearly beats all other heuristics for a wide range of values for p_1, including in those regions where the FC algorithm has its largest computational effort in combination with any heuristic. It is always possible, however, that by using either Kappa or Rho, a better result can be obtained for a particular instance, but let us recall that the result reported here in the GA case, is an average over a set of instances for each value of p_1. When observing results on experiments for *step*=2, Kappa, in general, has better performance than Rho and the GA. Nevertheless, the GA presents a reasonable performance over these instances, caused by the inclusion of the constraint density in the fitness function.

In order to support our study, statistical tests were run to validate the results. Despite of this, one may wonder about the overall performance of our strategy given that the computational cost of the GA is naturally higher given his population-based approach. It is then interesting to explore the trade-off between the gain in the number of constraint checks produced by the FC algorithm against the computational cost by any of the heuristics used including our approach. Results confirm the outcome on the previous experimentation. For



(a) non-uniform p_2, $step = 1$

Figure 4. Results on problems $\langle 20, 10\rangle$ with non-uniform p_2.

instance for less dense graphs with $p_1 = 0.5$ the best heuristic is Bz, and the GA has to work 140% more. But for dense graphs with $p_1 = 1$ the GA clearly generates better results on the number of constraint checks, with an additional cost of only 34% with respect to the effort taken by the Kappa heuristic. We think that with additional refinement of the FC-GA combination this percentage can be reduced, but this work is contemplated in future extensions of this investigation.

4 Conclusions

This article has proposed an innovative approach for using a GA to generate a dynamic variable ordering when solving CSPs. Using this scheme, under certain configuration of the GA, results are efficient, in terms of consistency checks. After testing for different values of parameter *step* (the number of variables to be instantiated before calling the GA again), it was found that best performance is shown when *step*=1. By establishing *step*=1, the fitness function in the GA could be seen as a deterministic heuristic that can be used to evaluate each of the uninstantiated variables and select that variable which maximizes the measure. This can be used as a single heuristic without considering the GA and probably would obtain as good results or better than those provided by the GA.

The FC-GA combination shows in general better results than the other heuristics, especially for highly constrained problems. It was also observed that when the probability p_2 is not uniform, the GA has a very competitive performance, achieving in some cases much better results than the other heuristics. The

success of the GA in these cases is that the fitness function takes into account the degree of a variable.

Acknowledgments

This research was supported by ITESM under the Research Chair CAT-010 and the CONACyT Project under grant 41515.

References

1. E. Tsang. *Foundations of Constraint Satisfaction.* Computation in Cognitive Science. Academic Press, London, 1993.
2. V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine 13(1):32-44*, 1992.
3. R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68(2):211-242, 1994.
4. P.W. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117-133, 1983.
5. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Alabama, 1989.
6. B. G. W. Craenen, A.E. Eiben, and J.I. van Hemert. Comparing evolutionary algorithms for binary constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 7(5):424–444, 2003.
7. A. E. Eiben. Evolutionary algorithms and constraint satisfaction: Definitions, survey, methodology and research directions. pages 13–58, 2001.
8. P. Prosser. Binary constraint satisfaction problems: Some are harder than others. In *Proceedings of the European Conference in Artificial Intelligence*, pages 95–99, Amsterdam, Holland, 1994.
9. B.M. Smith. Phase transition and the mushy region in constraint satisfaction problems. In *Proceedings of the European conference in Artificial Intelligence*, pages 100–104, Amsterdam, Holland, 1994.
10. B. M. Smith. Constructing an asymptotic phase transition in binary constraint satisfaction problems. *Journal of Theoretical Computer Science (Issue on NP-Hardness and Phase Transitions)*, 265(1):265–283, 2001.
11. I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In American Association for Artificial Intelligence, editor, *In Proceedings af AAAI-96*, 1996.
12. I.P. Gent, E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proceedings of CP-96*, pages 179–193. Springer, 1996.