An Experimental Comparison of SAWing EAs for a new Class of Random Binary CSPs

B.G.W. Craenen and A.E. Eiben
Vrije Universiteit
Faculty of Exact Sciences
De Boelelaan 1081
1081 HV Amsterdam
The Netherlands

Abstract - Evolutionary approaches to constraint satisfaction problems (CSPs) are often tested on a set of randomly generated instances. Recently it has been shown that the frequently used random problem instance generators are not good enough. This implies that conclusions based on the usage of these generators need to be revised. In this paper, we perform an extensive experimental comparison of CSP solving EAs based on a new, improved generator. In particular, we compare two representations, two evaluation functions and EAs with and without the so-called SAWing mechanism, each of these with different population sizes and variation operators. The resulting systematic overview confirms some "myths", while refutes others. Most notably, the role of the population size and that of SAWing itself turn out to be different from what is usually assumed.

I. INTRODUCTION

CSPs form a challenging application area for evolutionary algorithms. The challenge comes from two considerations. First, CSPs are in general computationally intractable and therefore form a general challenge for designers of algorithms. Second, in their original formulation CSPs have nothing to be optimised (see definitions in the next section). This makes EAs look inapplicable to tackle such problems. The last couple of years it has been shown that EA are very well capable of solving CSPs, e.g. graph colouring problems, 3-SAT problems, or randomly generated instances. One of the offered evolutionary approaches to tackling CSPs is formed by the so-called SAWing EAs, cf. section IV.

In this paper we attempt to perform a thorough and systematic experimental comparison of such algorithms, varying a number of key features and relating the performance differences to these variations. The rest of this paper is organised as follows. In section II we outline the basic notions of CSPs and in section III we describe the new CSP generator. In section IV we discuss penalty based approaches to evolutionary constraint satisfaction. The description of experiments and algorithm setups is given in section V, followed by a summary of results in section VI. Finally, section VII presents the conclusions.

II. CSPS

A *constraint network* consists of a set of variables $X_1, \ldots, X_n$ with respective domains $D_1, \ldots, D_n$, and a set of constraints $\mathcal{C}$. The Cartesian product of sets $D_1 \times \cdots \times D_n$ is called the *search space* and denoted by S. For $2 \leq k \leq n$, a constraint $c_{j_1,\ldots,j_k} \in \mathcal{C}, j = 1, \ldots, m$ is a subset of $D_{j_1} \times \cdots \times D_{j_k}$, where the $j_1, \ldots, j_k$ are distinct. We say that $c_{j_1,\ldots,j_k}$ is of arity k and that it bounds the variables $X_{j_1}, \ldots, X_{j_k}$ and that $\mathcal{C}^{j_1,\ldots,j_k}$ is the set of constraints that bound variables $X_1, \ldots, X_k$. For convenience, we use the shorthands c_j and $\mathcal{C}^j$ if this cannot lead to confusion. For a given constraint network, the *Constraint Satisfaction Problem* (CSP) asks for all the n-tuples $(d_1, \ldots, d_n)$ of values such that $d_i \in D_i$, $i = 1, \ldots, n$, and for every $c_{j_1,\ldots,j_k} \in \mathcal{C}$, $(d_{j_1}, \ldots, d_{j_k}) \notin c_{j_1,\ldots,j_k}, j = 1, \ldots, m$. Such an n-tuple $s \in S$ is called a *solution* of the CSP.

For an instance Π of CSP with n variables, its *constraint hypergraph* G^Π has n vertices $v_1, \ldots, v_n$, which correspond to the variables of Π and it contains a hyperedge $\{v_{j_1}, \ldots, v_{j_k}\}$ if and only if there exists a constraint of arity k that bounds the variables $X_{j_1}, \ldots, X_{j_k}$. The following convenient graph-theoretic representation of a CSP instance Π will be used; the incompatibility hypergraph of Π, C^Π, is an n-partite hypergraph of which the ith part corresponds to variable X_i of Π which has exactly $|D_i|$ vertices, one for each value in D_i. In C^Π there exists a hyperedge $\{v_{j_1}, \ldots, v_{j_k}\}$, if and only if the corresponding values $d_{j_1} \in D_{j_1}$, $d_{j_2} \in D_{j_2}, \ldots, d_{j_k} \in D_{j_k}$ are in (not allowed by) some constraint that bounds the corresponding variables. Hence, the decision version of CSP is equivalent to asking if there exists a set of vertices in C containing exactly one vertex from each part while not 'containing' any hyperedge, i.e., if there exists an independent set with one vertex from each part[1].

Note that for the sake of simplicity we study binary

[1] The superscript from both the constraint and the incompatibility hypergraph will be omitted when it is clear from the context what instance is referred too

CSPs where all constraints have arity $k = 2$ (bound two variables) and where all the variable domains contain the same number of values D. We adhere to this simplification because it restricts experimental complexity and every CSP of arity larger than two has an equivalent binary CSP ([6]).

III. CSP GENERATORS

In [1], Achlioptas et al. show that the so-called Models A to D (defined below) are unsuitable for the study of phase transition and threshold phenomena such as CSPs. This is because the instances they asymptotically generate have almost certainly no solutions. A general framework for these models, presented in [4], [5] generates instances in two steps:

Step 1: Either (i) each one of the $\binom{n}{2}$ edges is selected to be in G independently of all other edges with probability p_1 (*constraint density*), or (ii) we uniformly select a random set of edges of size $p_1\binom{n}{2}$.

Step 2: Either (i) for every edge of G each one of the D^2 edges in C is selected with probability p_2 (*constraint tightness*), or (ii) for every edge of G we uniformly select a random set of edges in C of size $p_2 D^2$

Combining the options for the two sets, we get four slightly different models for generating random CSPs, in particular, in the terminology used in [5], if both Step 1 and 2 are done with option (i), we get Model A, while if both steps are done with option (ii), we get Model B.

As Achlioptas et al. show in [1] Model A generates almost certainly unsatisfiable instances for every $p_2 \neq 0$, while Model B generates almost certainly unsatisfiable instances for every $p_2 \geq 1/D$ (analogously for the other two models). In the same paper an alternative model for generating random CSP instances is proposed. The new model (Model E) does not suffer from the deficiencies underlying the other models. This model resembles the model used for generating random Boolean formulas for the satisfiability problem and the constraints it generates are similar to the 'nogoods' proposed by Williams and Hogg ([7]). This model is defined as:

Definition 1: C^{Π} is a random n-partite graph with D vertices in each part constructed by uniformly, independently and with repetitions selecting $m = p\binom{n}{k}D^k$ hyperedges out of the $\binom{n}{k}D^k$ possible ones, with $k = 2$ for binary constraint networks. Also, let $r = m/n$ denote the ratio of the selected edges to the number of variables.

Such a model can be fully specified as $\mathrm{E}(n, m, D, k)$, where n is the number of variables, m is the number of constraints, D is the number of values in each domain and k is the arity of each constraint. Informally one could say that Model E works by choosing uniformly, independently and with repetitions conflicts between two values of two different variables.

It was known for Model A to D, that, when one of their parameters was varied, the generated CSP would exhibit a so called *phase transition*, where problems change from being relatively easy to solve to being very easy to prove unsolvable. The region where the probability that a problem is soluble changes from almost zero to almost one is generally indicated as the *mushy region*. In the mushy region, problems are in general difficult to solve or prove unsolvable and therefore of particular interest when comparing different algorithms for efficiency. Model E also exhibits a phase transition when one of its variables is changed and there are bounding formulas for the mushy region. In this paper, all CSP instances are generated using Model E with $n = 15$ variables, domain size $D = 15$, $k = 2$, probabilities from the set $\{0.20, 0.22, 0.24, 0.26, 0.28, 0.30, 0.32, 0.34, 0.36, 0.38\}$, and the corresponding values of m (see Definition 1). This puts all generated instances in the mushy region.

IV. PENALTIES AND THEIR ADAPTATION

In principle, every penalty function is an (heuristic) estimate of the badness of a given candidate solution. Such heuristics usually try to estimate the distance to feasible solution, or the costs of repairing the unfeasibility. Two basic types of estimations, thus penalty functions, are used very often: penalty for violated constraints and, penalty for wrongly instantiated variables.

For a formal description let us assume that we have constraints c_i $(i \in \{1, \ldots, m\})$, variables v_j $(j \in \{1, \ldots, n\})$, and let C^i be the set of constraints involving variable v_i. Then the penalties relative to the two options above described can be expressed as follows:

$$f_1(s) = \sum_{i=1}^{m} w_i \times \chi(s, c_i), \tag{1}$$

where

$$\chi(s, c_i) = \begin{cases} 1 & \text{if } s \text{ violates } c_i \\ 0 & \text{otherwise} \end{cases}$$

respectively

$$f_2(s) = \sum_{i=1}^{n} w_i \times \chi(s, C^i), \tag{2}$$

where

$$\chi(s, C^i) = \begin{cases} 1 & \text{if } s \text{ violates at least one } c \in C^i \\ 0 & \text{otherwise} \end{cases}$$

Obviously, for the above functions f_1, f_2 and for each $s \in S$ we have that s is a solution if and only if $f_i(s) = 0$. These definitions are independent from any given representation.

The Stepwise Adaptation of Weights (SAW) as a general mechanism to handle constraints in an EA, has been introduced by Eiben and van der Hauw [3]. The original

Expr.nr.	Fitness	Representation	Pop.size	Crossover
1	Variable + SAW	Integer	1	
2,3,4			10	1 point, uniform, 4 parent diagonal
5,6,7			100	1 point, uniform, 4 parent diagonal
8	Variable + SAW	Permutation	1	
9,10,11			10	Order1, Cycle, Pmx
12,13,14			100	Order1, Cycle, Pmx
15	Constraint + SAW	Integer	1	
16,17,18			10	1 point, uniform, 4 parent diagonal
19,20,21			100	1 point, uniform, 4 parent diagonal
22	Constraint + SAW	Permutation	1	
23,24,25			10	Order1, Cycle, Pmx
26,27,28			100	Order1, Cycle, Pmx
29	Variable, no SAW	Integer	1	
30,31,32			10	1 point, uniform, 4 parent diagonal
33,34,35			100	1 point, uniform, 4 parent diagonal
36	Variable, no SAW	Permutation	1	
37,38,39			10	Order1, Cycle, Pmx
40,41,42			100	Order1, Cycle, Pmx
43	Constr, no SAW	Integer	1	
44,45,46			10	1 point, uniform, 4 parent diagonal
47,48,49			100	1 point, uniform, 4 parent diagonal
50	Constr, no SAW	Permutation	1	
51,52,53			10	Order1, Cycle, Pmx
54,55,56			100	Order1, Cycle, Pmx

TABLE I

EXPERIMENTAL SETUP FOR THE DIFFERENT RUNS

idea behind the mechanism is that constraints that are not satisfied after a certain number of steps must be hard, thus must be given a high weight (penalty). This technique was successful in solving 3-SAT problems, graph 3-colouring problems and random binary CSPs, see [2] for an overview.

Technically, the SAW mechanism is an add-on to the above fitness function definitions amounting to revise the weights during a run, based on the status of the search process. This feature is mostly implemented by pausing the algorithm at regular intervals, e.g. after T_p fitness evaluations, and adding a constant increment Δw to each w_i that belongs to a violated constraint (or wrongly instantiated variable) in the best individual.

Setting the increment Δw to 1, the corresponding weight update mechanisms can be formally described as follows:

$$w_i \leftarrow w_i + \chi(X^*, c_i) \text{ for } i \in \{1, \ldots, m\} \text{ (SAW}_{con}\text{)}$$

respectively

$$w_i \leftarrow w_i + \chi(X^*, \mathcal{C}^i) \text{ for } i \in \{1, \ldots, n\} \text{ (SAW}_{var}\text{)}$$

with X^* denoting the best individual in the population found so far.

The algorithms reported in the literature used a fitness function of type 1 (see equation 1) for the 3-SAT and the random binary CSP application, and a fitness function of type 2 (see equation 2) for graph colouring. A remarkable feature of these algorithms is the very small populations, in some applications population size 1 turned out to be optimal.

V. EXPERIMENTS

In the present study we are investigating several aspects of constraint solving EAs. First, we are interested in the differences between algorithms using a fitness function of the first type (constraint-based) and the second type (variable-based). Second, we compare the effects of using two different representations. The integer representation is rather straightforward: each individual has length n (the number of variables) and the domain of the given variable forms the set of possible values for that position. In the order-based representation, an individual is a permutation of the variables and a simple decoder is used to assign domain values to each variable in the order they appear in a given permutation. The decoder sequentially takes variables from the permutation and tries to instantiate it with values that do not violate any constraints that bind already instantiated variables. If the decoder does not find such a value, the variable is left uninstantiated[2], note therefore that the decoder only searches the feasible portion of the search space. The third aspect we look at is the population size. The first SAWing EA applications suggested that

[2]Technically, it is instantiated to a special value indicating a conflict.

	Exp. 17 f_1, integer		Exp. 3 f_2, integer		Exp. 26 f_1, order-based		Exp. 12 f_2, order-based	
	SR	AES	SR	AES	SR	AES	SR	AES
0.20	0.8	1452.7	0.948	1053.59	1	100	1	100
0.22	0.62	2119.25	0.868	1635.91	1	100	1	100
0.24	0.328	1664.32	0.724	2825.83	1	100.52	1	100.52
0.26	0.176	1447.68	0.38	5026.57	1	114.136	1	112.328
0.28	0.068	3407.18	0.112	6553	1	232.808	1	232.096
0.30	0.016	1575	0.072	4328.11	1	966.52	1	1045.42
0.32	0.004	1492	0.012	4392	1	4036.66	0.988	3804.41
0.34	0	-	0.004	860	0.696	18836.3	0.688	14942.2
0.36	0	-	0	-	0.284	22861.6	0.296	19497.5
0.38	0	-	0	-	0.024	48761.7	0.02	49700.8

TABLE II

COMPARING CONSTRAINT AND VARIABLE BASED PENALTIES, f_1 VS. f_2, WITH SAWING

	Exp. 29 int. pop.size 1		Exp. 36 order b. pop.size 1		Exp. 33 int, pop.size 100		Exp. 40 order b. pop.size 100	
	SR	AES	SR	AES	SR	AES	SR	AES
0.2	1	408	1	8.912	1	2716.24	1	100
0.22	1	552.688	1	15.676	1	3519.84	1	100
0.24	1	866.604	1	37.012	1	4760.56	1	100.52
0.26	0.992	1345.85	1	82.024	0.972	8386.91	1	115.648
0.28	0.972	3429.6	1	209.412	0.828	12037.8	1	231.352
0.3	0.796	6098.46	1	799.608	0.576	17521.4	1	914.064
0.32	0.596	9886.1	1	3006.65	0.268	18338.5	0.988	3878.97
0.34	0.22	9140.64	0.8	12394.2	0.14	19656.6	0.66	12339
0.36	0.064	16395.3	0.32	13891.4	0.048	18953.3	0.24	14592.8
0.38	0.024	1819.33	0.024	14450.8	0.008	24270	0.012	50946

TABLE III

COMPARING INTEGER AND ORDER-BASED REPRESENTATION WITH f_2, NO SAW, POPULATION SIZE 1 AND POPULATION SIZE 100

a (1+1) scheme is a good heuristic setting. To check if this suggestion is still valid with the new problem instances, we run all algorithm variants with population sizes 1, 10 and 100. Fourth, we investigate the effects of the SAW mechanism itself. Although the SAW mechanism has been used in different applications and settings (i.e. fitness function and representation type), so far, no systematic investigation of all combinations of these settings has been published. Here we run all experiments with and without the weight adaptation mechanism on the new problem instances, meanwhile creating a comprehensive benchmark set for future work. Finally, we try a number of crossovers for those algorithms with a population size greater than 1. For the integer representation we try 1 point-, uniform-, 4 parent diagonal crossovers, while for the order-based representation we experiment with Order1, Cycle, and Pmx crossover. The complete overview of all experiments is given in table I.

There are also a number of shared features among our algorithm variants. The maximum number of fitness evaluations is always 100.000 and we use linear ranking selection with bias: b = 1.5. For the SAWing variants, the weight update period is set to $T_p = 250$ evaluations and $\Delta w = 1$ is used. The mutation operator for integer representation chooses a variable to be mutated and a new value for this variable uniform randomly. For the order-based algorithms we use a swap mutation operator.

As mentioned in section II, we use a set of 10 different probabilities for the Model E CSP generator. With each of these probabilities we generate 25 instances and perform 10 runs over each instance, resulting in 250 runs for every p value for each algorithm variant. Thus, one experiment, identified with a number in table I, consists of 2500 runs for the given algorithmic setup.

We use two measures of comparison for the algorithms. First the Success Rate (SR), indicating the percentage of the runs that were completed with a solution. Second, the Average number of Evaluations to a Solution (AES), together with its standard deviation. This measure is only defined when a solution was found. As a secondary measure AES is important as an indication of the speed of the algorithm.

VI. RESULTS

Space limitations do not allow us to present all experimental results here. Therefore, we restrict ourselves to

	Exp. 22 pop. size 1		Exp. 23 pop. size 10		Exp. 26 pop.size 100	
	SR	AES	SR	AES	SR	AES
0.2	1	8.912	1	10.552	1	100
0.22	1	15.676	1	13.088	1	100
0.24	1	37.656	1	26.784	1	100.52
0.26	0.968	74.5579	1	63.616	1	114.136
0.28	0.792	115.884	1	279.928	1	232.808
0.3	0.44	142.582	0.964	3781.09	1	966.52
0.32	0.24	201.217	0.74	13439.8	1	4036.66
0.34	0.076	193.053	0.352	15759	0.696	18836.3
0.36	0.012	223.33	0.108	17877	0.284	22861.6
0.38	0	-	0.004	4420	0.024	48761.7

TABLE IV

THE EFFECT OF POPULATION SIZE FOR ORDER-BASED REPRESENTATION USING CONSTRAINT BASED PENALTIES.

	Exp. 54 f_1, no SAW		Exp. 26 f_1, SAW		Exp. 40 f_2, no SAW		Exp. 12 f_2, SAW	
	SR	AES	SR	AES	SR	AES	SR	AES
0.2	1	100	1	100	1	100	1	100
0.22	1	100	1	100	1	100	1	100
0.24	1	100.52	1	100.52	1	100.52	1	100.52
0.26	1	115.648	1	114.136	1	115.648	1	112.328
0.28	1	231.352	1	232.808	1	231.352	1	232.096
0.3	1	914.064	1	966.52	1	914.064	1	1045.42
0.32	0.988	3878.97	1	4036.66	0.988	3878.97	0.988	3804.41
0.34	0.66	12339	0.696	18836.3	0.66	12339	0.688	14942.2
0.36	0.24	14592.8	0.284	22861.6	0.24	14592.8	0.296	19497.5
0.38	0.012	50946	0.024	48761.7	0.012	50946	0.02	49700.8

TABLE V

THE EFFECT OF SAWING FOR ORDER-BASED REPRESENTATION USING POPULATION SIZE 100 AND ORDER1 CROSSOVER.

presenting the outcomes that back-up our most interesting findings.

It seems to have little effect in changing between constraint- and variable-based penalty (fitness) functions. Without SAWing there is no difference at all while with SAWing and order-based representation the differences are small with a slight advantage of constraint based penalties (table II right two columns). With SAWing and integer representation, variable based penalties work better (table II left two columns).

As for the representation, it is clearly proven that the order-based representation (together with the decoder) is superior to the straightforward integer representation: higher SR, lower AES and standard deviation (not shown here), see table III.

The optimal population size is clearly related to the SAWing feature. For all algorithms using no SAWing, the optimal population size is 1. Nevertheless, for all algorithms using SAWing, the optimal population size is the largest one among those tested: 100. In table IV we give an illustration for order-based representation using constraint based penalties.

The effect of SAWing is again related to the representation and, to our surprise, also to the population size. In case of integer representation and order-based representation with small population sizes (1 or 10) it does not improve performance. For order-based representation and population size 100 it does so for both penalty schemes, cf. table V.

VII. CONCLUSIONS

The systematic and thorough experimentation reported in this paper confirms some "myths", while refutes others. The superiority of the order-based representation has been confirmed by all pairwise comparisons of corresponding algorithm variants. Apparently, the combination of a smaller search space (15! vs. 15^{15}) and the decoder – be it very simple – make the order-based EAs so powerful that no feature within the integer based variants can compensate their advantage. To this end, it is remarkable that for the easiest instances ($p \in \{0.2, \ldots, 0.26\}$) a solution is found in the initial population or the first generation (AES is around 100), meaning that (almost) no search is required – randomly generated chromosomes can be decoded to good solutions.

For illustration see table III. With population size 1 (experiment 36) it takes less than 100 generations

	Experiment 26, SAWing			Experiment 36, no SAWing		
	SR	AES	Std. Deviation	SR	AES	Std. Deviation
0.2	1	100	0	1	8.912	9.46778
0.22	1	100	0	1	15.676	15.6319
0.24	1	100.52	4.86294	1	37.012	42.9266
0.26	1	114.136	43.1245	1	82.024	77.4873
0.28	1	232.808	290.339	1	209.412	282.486
0.3	1	966.52	2344.56	1	799.608	2014.32
0.32	1	4036.66	7707.84	1	3006.65	5130.01
0.34	0.696	18836.3	26730.8	0.8	12394.2	18062.9
0.36	0.284	22861.6	27187.2	0.32	13891.4	18001.8
0.38	0.024	48761.7	35177.1	0.024	14450.8	8015.43

TABLE VI
COMPARING THE BEST SAWING AND NON SAWING ALGORITHM VARIANTS

(mutation-selection cycles) to find a solution. For population size 100 (experiment 40) a solution is found in either in the initial population (AES=100) or in the first population ($100 \leq$ AES ≤ 200). This suggests that the decoder delivers the greatest problem solving power, the role of evolutionary search seems to be secondary.

Somewhat surprisingly, we found only small differences between using penalties based on constraints (f_1) and variables (f_2). The number of constraints is much larger than the number of variables, therefore f_1 should carry more information, but this does not seem to be crucial.

Looking at the effects of the population size also yields some surprising facts. Contrary to previous suggestions in the literature ([3], [2]) the optimal population size for the SAWing EAs is larger than 1. Among the values tested here 10 was optimal for the integer representation and 100 for order-based representation. However, this observation does not hold for EAs without SAWing, where all variants performed best with the (1+1) setup: population size 1, no crossover, only mutation. Recall, that a SAWing EA is working on two tasks simultaneously: solving the given CSP and finding appropriate settings for the weights in the fitness function. It seems that this second task is performed better when using large populations.

As for the genetic operators we can conclude that uniform crossover is preferable for integer representation and order1 for order-based.

Finally, the comparisons of algorithm variants with and without SAWing also led to unexpected results. SAWing seems to offer an advantage only for the order-based EAs with a large population, cf. table V. Furthermore, the best algorithm without SAWing (experiment setup 36) is better than the best one with SAWing (experiment setup 26), see table VI. To this end, let us note that we did not optimise the parameters T_p and Δw of the SAWing procedure, just used previously suggested values.

The outcomes of this investigation should also be considered from the perspective of the used problem instances. We generated these instances by a recently proposed problem instance generator that eliminates serious deficiencies of generators used before. However, for the new generator, there are no published results in evolutionary computation yet. This study therefore also serves as a benchmark for future comparative research. The generator and also all problem instances we used can be downloaded from `www.cs.vu.nl/~bcraenen`.

Ongoing research is following the hints given by 3-SAT applications of SAWing EAs. In particular, $(1 + \lambda)$ schemes proved to be better than the simple (1+1) setup. Furthermore, the value of λ also had a clear effect on performance and in this setting SAWing was certainly advantageous. It is an interesting question whether such observations also hold on randomly generated binary CSPs.

References

[1] D. Achlioptas, L.M. Kirousis, E. Kranakis, D. Krizanc, M.S.O. Molloy, and Y.C. Stamatiou. Random constraint satisfaction: A more accurate picture. In G. Smolka, editor, *Principles and Practice of Constraint Programming — CP97*, number 1330 in Lecture Notes in Computer Science, pages 107–120, Berlin, 1997. Springer-Verlag.

[2] A.E. Eiben and J.I. van Hemert. SAW-ing EAs: Adapting the fitness function for solving constrainted problems. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 389–402. McGraw-Hill, 1999.

[3] A.E. Eiben and J.K. van der Hauw. Graph coloring with adaptive genetic algorithms. Technical report, Leiden University, November 1996.

[4] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Journal of Artificial Intelligence*, 81:81–109, 1996.

[5] B.M. Smith and M.E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Journal of Artificial Intelligence*, 81(1-2):155–181, 1996.

[6] E.P.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press Limited, 1993.

[7] C.P. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Journal of Artificial Intelligence*, 70:73–117, 1994.