Solving Binary Constraint Satisfaction Problems Using Evolutionary Algorithms with an Adaptive Fitness Function

A.E. Eiben[1,2], J.I. van Hemert[1], E. Marchiori[1,2] and A.G. Steenbeek[2]

[1] Dept. of Comp. Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, NL
[2] CWI, P.O. Box 94079, 1090 GB Amsterdam, NL

Abstract. This paper presents a comparative study of Evolutionary Algorithms (EAs) for Constraint Satisfaction Problems (CSPs). We focus on EAs where fitness is based on penalization of constraint violations and the penalties are adapted during the execution. Three different EAs based on this approach are implemented. For highly connected constraint networks, the results provide further empirical support to the theoretical prediction of the phase transition in binary CSPs.

1 Introduction

Evolutionary algorithms are usually considered to be ill-suited for solving constraint satisfaction problems. Namely, the traditional search operators (mutation and recombination) are 'blind' to the constraints, that is, parents satisfying a certain constraint may very well result in an offspring that violates it. Furthermore, while EAs have a 'basic instinct' to optimize, there is no objective function in a CSP – just a set of constraints to be satisfied. Despite such general arguments, in the last years there have been reports on quite a few EAs for solving CSPs having a satisfactory performance. Roughly speaking, these EAs can be divided into two categories: those based on exploiting heuristic information on the constraint network [6, 14, 21, 22], and those using a fitness function (penalty function) that is adapted during the search [2, 4, 5, 7, 9, 10, 17, 18]. In this paper we investigate three methods from the second category: the co-evolutionary method by Paredis [17], the heuristic-based microgenetic algorithm by Dozier et al [4], and the EA with stepwise adaptation of weights by Eiben et al. [10]. We implement three specific evolutionary algorithms based on the corresponding methods, called `COE`, `SAW`, and `MID`, respectively, and compare them on a test suite consisting of randomly generated binary CSPs with finite domains.

The results of the experiments are used to assess empirically the relative performance of the three different methods within the same category, thereby providing suggestions as to which implementation of the same general idea is the most promising. We use randomly generated problem instances for the experiments, where the hardness of the problem instances is influenced by two parameters: constraint density and constraint tightness. By running experiments on 25 different combinations of these parameters we gain detailed feedback on EA behavior and can validate theoretical predictions on the location of the

phase transition. In summary, on the 625 problem instances considered, MID performs better than the other two EAs with respect to the success rate (i.e., how many times a solution is found). The success rate of SAW is slightly worse on harder problem instances corresponding to higher constraint density and tightness, while the performance of COE is rather unsatisfactory when compared with the performance of one of the other two algorithms. Concerning the computational effort, it is worthwhile to note that SAW requires fewer fitness evaluations to find a solution than the other two EAs. The obtained results show that for higher constraint density and tightness, all three EAs are unable to find a solution. This behavior is in accordance with theoretical predictions on the phase transition in binary CSPs (cf. [23]).

The paper is organized as follows: the next section describes the notion of constrained problems and it deals more specifically with random binary CSPs. In Section 3 various evolutionary approaches for solving CSPs are discussed and the three EAs COE, SAW, and MID are introduced. In Section 4 the experimental setup is given, followed by the results. Finally, in Section 5 we summarize our conclusions and we give some hints on future work.

2 Constraint satisfaction problems

A *constraint satisfaction problem* (CSP) is a pair $\langle S, \phi \rangle$, where S is a free search space and ϕ is a formula (Boolean function on S). A *solution of a constraint satisfaction problem* is an $s \in S$ with $\phi(s) = true$. Usually a CSP is stated as a problem of finding an instantiation of variables $v_1, \ldots, v_n$ within the finite domains $D_1, \ldots, D_n$ such that constraints (relations) $c_1, \ldots, c_m$ hold. The formula ϕ is then the conjunction of the given constraints. One may be interested in one, some or all solutions, or only in the existence of a solution. We restrict our discussion to finding one solution.

More specifically, we consider binary constraint satisfaction problems over finite domains, where constraints act between pairs of variables. This is not restrictive since any CSP can be reduced to a binary CSP by means of a suitable transformation which involves the definition of more complex domains (cf. [24]). A class of random binary CSPs can be specified by means of four parameters $\langle n, m, d, t \rangle$, where n is the number of variables, m is the uniform domain size, d is the probability that a constraint exists between two variables, and t is the probability of a conflict between two values across a constraint. CSPs exhibit a *phase transition* when a parameter is varied. At the phase transition, problems change from being relatively easy to solve (i.e., almost all problems have many solutions) to being very easy to prove unsolvable (i.e., almost all problems have no solutions). The term *mushy region* is used to indicate that region where the probability that a problem is soluble changes from almost zero to almost one. Within the mushy region, problems are in general difficult to solve or to prove unsolvable. An important issue in the study of binary CSPs is to identify those problem instances which are very hard to solve [3]. Recent theoretical investigations ([23, 26]) allow one to predict where the hardest problem instances

should occur. Williams and Hogg in [26] develop a theory that predicts that the phase transition occurs when per variable there are a critical number of nogoods (i.e., of conflicts between that variable and all other ones)[3]. Smith in [23] conjectures that the phase transition occurs when problems have, on average, just one solution.

An experimental investigation with a complete algorithm (i.e., an algorithm that finds a solution or detects unsatisfiability) based on forward checking and on conflict-directed backjumping, is given by Prosser in [20], which provides empirical support to the theoretical prediction given in [23, 26] for higher density/tightness of the constraint networks. We will see that this trend is supported also by our experimental investigation on three specific evolutionary algorithms for CSPs. Being stochastic techniques, EAs are in general unable to detect inconsistency, so our analysis of hard instances for these algorithms will necessarily be incomplete. However, we will see that the success rate of the three EAs on the considered problems provides a neat indication of which (d, t) regions contain hard problems for these EAs, indicating a phase transition which is in accordance with the one identified using the theoretical prediction in [23, 26] for relatively high values of d and t.

3 Constraint handling by penalties in EAs

There are several ways to handle constraints in an EA. At a high conceptual level we can distinguish two cases, depending on whether they are handled *indirectly* or *directly*. Indirect constraint handling means that the constraints are incorporated in the fitness function f such that the optimality of f implies that the constraints are satisfied. Then the optimization power of the EA can be used to find a solution. By direct constraint handling here we mean that the constraints are left as they are and 'something' is done in the EA to enforce them. Some commonly used options are repair mechanisms, decoding algorithms and using special reproduction operators [8, 15].

In the case of indirect constraint handling a lot of different fitness functions can satisfy the above requirement. A common way of defining a suitable fitness function is based on using penalties, usually penalizing the violation of each constraint and making the fitness function (to be minimized) the sum of such penalties. In the simplest case, each constraint violation scores one penalty point, hence the fitness function is just counting the number of violated constraints. A weighted sum, however, allows more appropriate measurement of quality, for instance harder constraints can be given higher weights. This would give a relatively high reward when satisfying them, thus directing the EAs attention to such constraints. Natural as it may sound, this idea is not so trivial to implement. The two main problems are that 1) estimating constraint hardness a priori for setting the weight appropriately may require substantial problem specific knowledge or computational efforts, 2) the appropriate weights may change

[3] Note that the expected number of nogoods per variable is $dtm^2(n-1)$.

during the problem solving process. A possible treatment for both problems is to have the algorithm setting the weights (penalties) itself and re-adjusting the weights during the search process.

In this paper we investigate three evolutionary algorithms that represent constraints by penalties and update the penalty function during the run. We retrieve the specification of these EAs from the literature and maintain the original parameter settings, whenever possible. Therefore, here we describe only the main features of the algorithms and refer to the cited articles for further details.

The first EA we consider is a heuristic-based microgenetic algorithm introduced by Dozier et al in [4]. It is called microgenetic because it employs a small population. Moreover, it incorporates heuristics in the reproduction mechanism and in the fitness function in order to direct the search towards better individuals. More precisely, the EA we implement works on a pool of 8 individuals. It uses a roulette-wheel based selection mechanism, and the steady state reproduction mechanism where at each generation an offspring is created by mutating a specific gene of the selected chromosome, called pivot gene, and that offspring replaces the worse individual of the actual population. Roughly, the fitness function of a chromosome is determined by adding a suitable penalty term to the number of constraint violations the chromosome is involved in. The penalty term depends on the set of breakouts[4] whose values occur in the chromosome. The set of breakouts is initially empty and it is modified during the execution by increasing the weights of breakouts and by adding new breakouts according to the technique used in the Iterative Descent Method ([16]). Therefore we have named this algorithm `MID`, standing for Microgenetic Iterative Descent.

The basic concept behind the *co-evolutionary approach* is the idea of having two populations constantly in battle with each other. This approach has been tested by Paredis on different problems, such as neural net learning [18], constraint satisfaction [17, 18] and searching for cellular automata that solve the density classification task [19]. The evolutionary algorithm used in the experiments, denoted as `COE`, is a steady-state EA, it has two populations, one is called the *solution population* and the other is called the *constraint population.* The fitness of an individual in either of these populations is based on a history of encounters. An *encounter* means that an individual from the constraint population is matched with an individual from the solution population. If the constraint is not violated by the solution, the individual from the solution population gets a point. If the constraint is violated the individual from the constraint population gets a point. The fitness of an individual is the amount of points it has obtained in the last 25 encounters. Every generation of the EA, 20 encounters are executed by repeatedly selecting an individual from each population. Then two parents are selected using linear ranked selection, with a bias of 1.5, as described by Whitley [25]. The two parents are crossed using a two-point reduced surrogate parents crossover, this makes sure that the children are different when

[4] A breakout consists of two parts: 1) a pair of values that violates a constraint; 2) a weight associated to that pair.

the parents differ. The two resulting children are then mutated using adaptive mutation. This means every allele has a chance of 0.001 of mutating, unless the two children are the same then the chance is increased to 0.01. The size of the constraint population is determined by the amount of constraints in the problem, the solution population however has a fixed size of 50 individuals.

The third EA we are studying is the so-called SAW-ing EA. The Stepwise Adaptation of Weights (SAW) mechanism has been introduced by Eiben and van der Hauw [9] as an improved version of the weight adaptation mechanism of Eiben, Raué and Ruttkay [6, 7]. In several comparisons the SAW-ing EA proved to be a superior technique for solving specific CSPs [1, 10]. The basic idea behind the SAW-ing 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). The realization of this idea constitutes of initializing the weights at 1 and re-setting them by adding a value Δw after a certain period. Re-setting is only applied to those constraints that are violated by the best individual of the given population. Earlier studies indicated the good performance of a simple (1+1) scheme, using a singleton population and exclusively mutation to create offspring. The representation is based on a permutation of the problem variables; a permutation is transformed to a partial instantiation by a simple decoder that considers the variables in the order they occur in the chromosome and assigns the first possible domain value to that variable. If no value is possible without introducing constraint violation, the variable is left uninstantiated. Uninstantiated variables are, then, penalized and the fitness of the chromosome (a permutation) is the total of these penalties. Let us note that penalizing uninstantiated variables is a much rougher estimation of solution quality than penalizing violated constraints. Yet, this option worked well for graph coloring, therefore we test it here without much modification.

4 Experimental Setup and Results

To generate a test suite we have used a problem instance generator that was loosely based on the generator of Gerry Dozier [2]. The generator first calculates the number of constraints that will be produced using the equation $\frac{n(n-1)}{2} \cdot d$. It then starts producing constraints by randomly choosing two variables and assigning a constraint between them. When a constraint is assigned between variable v_i and v_j, a table of conflicting values is generated. To produce a conflict two values are chosen randomly, one for the first and one for the second variable. When no conflict is present between the two values for the variables, a conflict is produced. The number of conflicts in this table is determined in advance by the equation $m(v_i) \cdot m(v_j) \cdot t$ where $m(v_i)$ is the domain size of variable i.

We have considered random binary constraints with $n = 15$ variables and uniform domains of $m = 15$ elements. These values are common in the empirical study of (random) CSPs. Later on we will discuss the effect of the varying of the number of variables. Each algorithm has been tested on the same 625 problem instances: for each combination of density d and tightness t (25 in total), we have

generated 25 instances and executed 10 independent runs on each instance. The algorithm performance is evaluated by two measures. The *Success Rate* (SR) is the percentage of instances where a solution has been found. The *Average number of Evaluations to Solution* (AES) is the number of fitness evaluations, i.e. the number of newly generated candidate solutions, in *successful* runs. Note, that if a run did not find a solution, the number of steps to a solution is not defined, consequently if for a certain combination of d and t SR = 0, then the AES is not defined. The specific details of the three EAs used in this comparison are mentioned in the previous section. Recall, that each variant is allowed to generate 100000 candidate solutions, i.e., the algorithms terminate if a solution is found or the limit of 100000 is reached.

density	alg.	tightness				
		0.1	0.3	0.5	0.7	0.9
0.1	COE	1.00 (3)	1.00 (15)	1.00 (449)	1.00 (2789)	0.62 (30852)
	MID	1.00 (1)	1.00 (4)	1.00 (21)	1.00 (87)	0.96 (2923)
	SAW	1.00 (1)	1.00 (1)	1.00 (2)	1.00 (9)	0.64 (1159)
0.3	COE	1.00 (96)	1.00 (11778)	0.18 (43217)	0.00 (-)	0.00 (-)
	MID	1.00 (3)	1.00 (50)	1.00 (323)	0.52 (32412)	0.00 (-)
	SAW	1.00 (1)	1.00 (2)	1.00 (36)	0.23 (21281)	0.00 (-)
0.5	COE	1.00 (1547)	0.08 (39679)	0.00 (-)	0.00 (-)	0.00 (-)
	MID	1.00 (10)	1.00 (177)	0.90 (26792)	0.00 (-)	0.00 (-)
	SAW	1.00 (1)	1.00 (8)	0.74 (10722)	0.00 (-)	0.00 (-)
0.7	COE	1.00 (9056)	0.00 (-)	0.00 (-)	0.00 (-)	0.00 (-)
	MID	1.00 (20)	1.00 (604)	0.00 (-)	0.00 (-)	0.00 (-)
	SAW	1.00 (1)	1.00 (73)	0.00 (-)	0.00 (-)	0.00 (-)
0.9	COE	0.912 (28427)	0.00 (-)	0.00 (-)	0.00 (-)	0.00 (-)
	MID	1.00 (33)	1.00 (8136)	0.00 (-)	0.00 (-)	0.00 (-)
	SAW	1.00 (1)	1.00 (3848)	0.00 (-)	0.00 (-)	0.00 (-)

Table 1. Sucess rates and the corresponding AES values (within brackets) for the co-evolutionary GA (`COE`), the Micro-genetic algorithm with Iterative Descent (`MID`), and the SAW-ing GA (`SAW`)

Table 1 summarizes the results of our experiments. Considering the success rate it is clear that `MID` performs equally or better than the other two EAs in all classes of instances. `SAW` has a lower SR than `MID` on harder problem instances, namely for the classes $(d = 0.1, t = 0.9)$, $(d = 0.3, t = 0.7)$, and $(d = 0.5, t = 0.5)$, but on two of these three ($(d = 0.1, t = 0.9)$ and $(d = 0.5, t = 0.5)$) it is more than twice as fast as `MID`, while being only approximately 30% less successful. The performance of `COE` is rather unsatisfactory also in relatively 'easy' classes, like for example the one characterized by $(d = 0.7, t = 0.3)$.

These observations can be explained by observing that MID employs a strong heuristic technique based on hill-climbing *and* an adaptive fitness function, while SAW and COE try to bias the search towards harder constraints in a more naive way. Recall that the version of SAW we are testing here considers only the uninstantiated variables, not the violated constraints, as the basis of the fitness function. This gives a very rough quality estimate as compared to the constraint based penalties of MID, i.e. using n variables instead of $\frac{n(n-1)}{2} \cdot d$ constraints. In this light, the exhibited performance of SAW can be seen as remarkably good. The method used in COE for dealing with harder constraints does not prove to be very effective. Concerning the computational effort (AES), it is worth noting that SAW usually requires much less evaluations to find a solution than the other two EAs. A possible reason for this fact could lay in the decoding mechanism SAW is using. Simple and unbiased as this decoder may seem, it could represent a successful heuristic.

The results of the experiments reported in Table 1 consider $n = 15$ variables and $m = 15$ domain size. It is interesting to investigate how the results scale up when we vary the number of variables n. The question of scaling up is interesting already for its own sake, but here we also have a special reason to look at it. In particular, measuring the performance of EAs by the AES is not as unbiased as it may look at the first glance. Namely, the use of the hill-climbing heuristic in MID, and the efforts spend on decoding in SAW are important for their good performance, but are invisible for the measure AES. In order to obtain a more fair comparison, we look at the steepness of the AES curve when increasing the problem size (n). We do not consider the CPU times, since it is much dependent on implementational details, network load, etc. Figure 1 illustrates how the performance of MID and SAW is affected by increasing n, when the other parameters are set to $m = 15$, $d = 0.3$, and $t = 0.3$. (Note that we do not consider COE because of its poor performance.) We consider values of n ranging from 10 till 40 with step 5 and observe that increasing the number of variables does not affect the success rates in this range of n values. The number of iterations that are needed in order to find a solution, however, is heavily affected and for both algorithms it exhibits a super-linear growth. Recall, that the exact heights of the data points on the curves are not relevant, it is the growth rate we are looking at. The two curves are similar in this respect, although up to $n = 35$ SAW is growing at a visibly slower rate. However, since the two curves are crossing at the end, we do not want to suggest a better scale-up behavior for either algorithms.

Considering the results in Table 1 from the point of view of the problem instances, we can observe success rates SR = 1 in the upper left corner, while SR = 0 is typical in the lower right corner, separated by a 'diagonal' indicating the mushy region. Technically, for higher constraint density and tightness, all three EAs are unable to find any solution. This is not surprising, because higher density and tightness yield problems that are almost always unsatisfiable. An other interesting aspect of the behavior of these algorithms is for which problem instances their performance rapidly degrade. The most difficult CSPs seem to

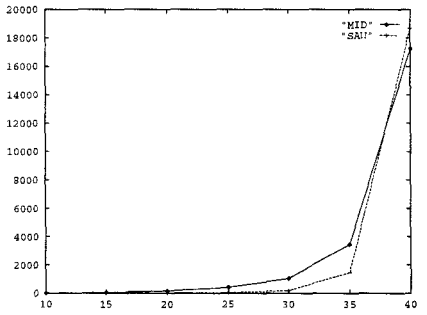

Fig. 1. Scale-up values for the SAW and MID algorithms.

start in the classes where $d \geq 0.3$ and $t = 0.7$, and where $d \geq 0.5$ and $t = 0.5$. These results are in accordance with theoretical predictions of the phase transition for binary CSP problems ([23, 26]). In fact, according to e.g. Smith prediction, for $n = m = 15$, the phase transition for binary CSPs is located to $t_{crit} = 0.725$ for $d = 0.3$, and to $t_{crit} = 0.539$ for $d = 0.5$.

We conclude this section with some observations on the behavior of the fitness function during the execution of these EAs. Figure 2 shows one typical run of each of the two winning methods SAW and MID, plotting the fitness of the best individual in the population during a run. The course of the fitness function in a typical run of MID suggests that the in the first generations a relatively good solution is found, where only few constraints remain to be satisfied. Then the fitness starts to go up and down by the adaptive mechanism on breakouts, and finally it jumps to zero (a solution). For SAW we see the penalty growing in the beginning, because of the increasing weights. Seemingly things get only worse, but then a solution is reached in just a few steps. A plausible interpretation of this behavior is that the algorithm is first looking for well-suited weights that make the problem 'easy', and solves the resulting problem (that is optimizes the resulting 'easy' function) afterwards.

5 Conclusion

In this paper we have performed a comparative experimental study on three EAs for solving CSPs, which have the common feature of employing an adaptive fitness function in order to direct the search.

Current work concerns investigating the performance of SAW if the same representation is used as in MID, i.e. integers instead of permutations, and if the fitness function is based on constraint violations instead of uninstantiated variables. Recall, that MID keeps the penalty term on violated constraints identical during the search and is adapting the penalty term on breakouts only. SAW and MID can thus be easily combined, if SAW-ing is applied to the first penalty term

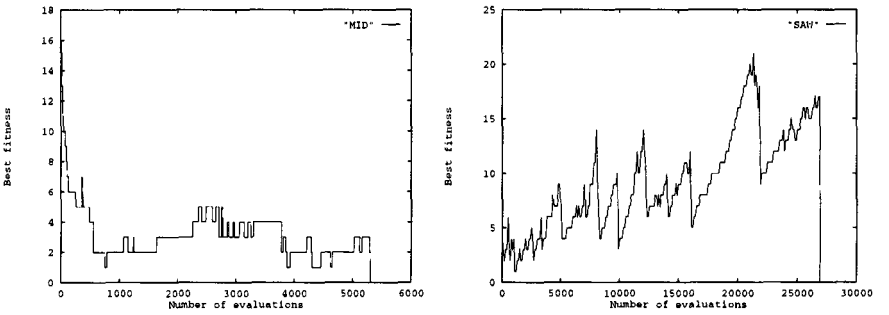

Fig. 2. Fitness (to be minimized) of the best individual in the population during a run of MID (left) and SAW (right).

and the MID mechanism is used to tune the breakouts during the run.

Future work will also involve comparison of EAs of the second group mentioned in the Introduction, those based on using information on the given constraint network.

References

1. Th. Bäck, A.E. Eiben, and M.E. Vink. A superior evolutionary algorithm for 3-SAT. In D. Waagen N. Saravanan and A.E. Eiben, editors, *Proceedings of the 7th Annual Conference on Evolutionary Programming*, Lecture Notes in Computer Science. Springer, 1998. in press.
2. J. Bowen and G. Dozier. Solving constraint satisfaction problems using a genetic/systematic search hybride that realizes when to quit. In Eshelman [11], pages 122–129.
3. P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Conference on Artificial Intelligence*, pages 331–337. Morgan Kaufmann, 1991.
4. G. Dozier, J. Bowen, and D. Bahler. Solving small and large constraint satisfaction problems using a heuristic-based microgenetic algorithms. In IEEE [12], pages 306–311.
5. G. Dozier, J. Bowen, and D. Bahler. Solving randomly generated constraint satisfaction problems using a micro-evolutionary hybrid that evolves a population of hill-climbers. In *Proceedings of the 2nd IEEE Conference on Evolutionary Computation*, pages 614–619. IEEE Press, 1995.
6. A.E. Eiben, P.-E. Raué, and Zs. Ruttkay. Constrained problems. In L. Chambers, editor, *Practical Handbook of Genetic Algorithms*, pages 307–365. CRC Press, 1995.
7. A.E. Eiben and Zs. Ruttkay. Self-adaptivity for constraint satisfaction: Learning penalty functions. In *Proceedings of the 3rd IEEE Conference on Evolutionary Computation*, pages 258–261. IEEE Press, 1996.
8. A.E. Eiben and Zs. Ruttkay. Constraint satisfaction problems. In Th. Bäck, D. Fogel, and M. Michalewicz, editors, *Handbook of Evolutionary Algorithms*, pages **C5.7:1–C5.7:8. IOP Publishing Ltd. and Oxford University Press, 1997.**

9. A.E. Eiben and J.K. van der Hauw. Adaptive penalties for evolutionary graph-coloring. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution'97*, number 1363 in LNCS, pages 95–106. Springer, Berlin, 1997.
10. A.E. Eiben, J.K. van der Hauw, and J.I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4:25–46, 1998.
11. L.J. Eshelman, editor. *Proceedings of the 6th International Conference on Genetic Algorithms.* Morgan Kaufmann, 1995.
12. *Proceedings of the 1st IEEE Conference on Evolutionary Computation.* IEEE Press, 1994.
13. *Proceedings of the 4th IEEE Conference on Evolutionary Computation.* IEEE Press, 1997.
14. E. Marchiori. Combining constraint processing and genetic algorithms for constraint satisfaction problems. In Th. Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 330–337. Morgan Kaufmann, 1997.
15. Z. Michalewicz and M. Michalewicz. Pro-life versus pro-choice strategies in evolutionary computation techniques. In Palaniswami M., Attikiouzel Y., Marks R.J., Fogel D., and Fukuda T., editors, *Computational Intelligence: A Dynamic System Perspective*, pages 137–151. IEEE Press, 1995.
16. P. Morris. The breakout method for escaping from local minima. In *Proceedings of the 11th National Conference on Artificial Intelligence, AAAI-93*, pages 40–45. AAAI Press/The MIT Press, 1993.
17. J. Paredis. Co-evolutionary constraint satisfaction. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, number 866 in Lecture Notes in Computer Science, pages 46–56. Springer-Verlag, 1994.
18. J. Paredis. Co-evolutionary computation. *Artificial Life*, 2(4):355–375, 1995.
19. J. Paredis. Coevolving cellular automata: Be aware of the red queen. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, San Francisco, CA, 1997. Morgan Kaufmann.
20. P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109, 1996.
21. M.C. Riff-Rojas. Using the knowledge of the constraint network to design an evolutionary algorithm that solves CSP. In IEEE [13], pages 279–284.
22. M.C. Riff-Rojas. Evolutionary search guided by the constraint network to solve CSP. In IEEE [13], pages 337–348.
23. B.M. Smith. Phase transition and the mushy region in constraint satisfaction problems. In A. G. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 100–104. Wiley, 1994.
24. E. Tsang. *Foundation of Constraint Satisfaction.* Academic Press, 1993.
25. D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 116–123, San Mateo, California, 1989. Morgan Kaufmann Publishers, Inc.
26. C.P. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.