# Stepwise Adaption of Weights with Refinement and Decay on Constraint Satisfaction Problems

**B.G.W. Craenen**
Computational Intelligence Group
Faculty of Exact Sciences
Vrije Universiteit Amsterdam
bcraenen@cs.vu.nl

**A.E. Eiben**
Computational Intelligence Group
Faculty of Exact Sciences
Vrije Universiteit Amsterdam
gusz@cs.vu.nl

## Abstract

Adaptive fitness functions have led to very successful evolutionary algorithms (EA) for various types of constraint satisfaction problems (CSPs). In this paper we consider one particular fitness function adaptation mechanism, the so called Stepwise Adaption of Weights (SAW). We compare algorithm variants including two penalty systems and we experiment with extensions of the SAW mechanism utilizing a refinement function and a decay function. Experiments are executed on binary CSP instances generated by a recently proposed method (method E). This new method for generating problem instances allows one single hardness parameter and is well suited to study algorithmic behavior around the phase transition. The results show that the original version of the SAW mechanism is very robust and has a comparable or better performance than the extended SAW mechanisms.

## 1 INTRODUCTION

Informally, a constraint satisfaction problem (CSP) consists of finding an assignment of values to variables in such a way that the restrictions imposed by the given set of constraints are satisfied. Constraint satisfaction is a fundamental topic in artificial intelligence with great practical and theoretical relevance. On the practical side, CSPs have relevant applications in planning, default reasoning, scheduling, etc. and a great deal of practical problems are constrained. Theoretically, CSPs are, in general, computationally intractable (NP-hard) thereby forming a big challenge to algorithm designers.

Evolutionary algorithms are known for their good performance in the field of optimization. Constraint handling, however, is not straightforward in an EA as the traditional search operators, mutation and recombination do not heed constraints (or their violation). Nevertheless, there is a growing body of literature on applying EAs to various CSPs, such as graph-coloring, satisfiability, or randomly generated binary CSPs. One research line is based on a pure penalty approach, where the evolutionary algorithm handles all constraints indirectly. That is, *all* constraint violations are turned into penalties and the EA is “only” optimizing an unconstrained problem where the fitness function is composed from these penalties. In this approach any direct constraint handling (e.g. specific constraint respecting mutation or crossover operators, or repair mechanisms fixing some constraint violations) is absent, making it very transparent and general.

The SAW procedure is an add-on to this general scheme to boost performance (to minimize the penalties more effectively): It adaptively changes the composition of the fitness function during the search process. Here we experiment with the SAW EA with two penalty systems (penalizing the violated constraints vs. penalizing the wrongly instantiated variables), different refinement functions, and decay functions, so as to give a complete overview of different variations on this algorithm.

Another novelty in the present work is the usage of a recently proposed problem instance generator. As shown by Achlioptas et al. in [1], the widely applied CSP generators (much used in EA research) have serious deficiencies. Most importantly, the generated instances tend to be asymptotically unsolvable, preventing a sound study of algorithmic behavior around the phase transition. The proposed alternative cures this problem and has an additional nice feature: It allows one single hardness parameter, making the presentation (and interpretation) of results much easier.

The paper is organized as follows. Section 2 will define the notation and terminology we use for describing CSPs and in section 4 we will examine further how CSPs are generated using model E. In section 4.1 we will discuss how the standard stepwise adaption of weights method will work on these CSPs while in section 4.2 the refining function and the adaptation mechanisms are discussed. In section 4.3 the decay mechanisms are discussed, followed by the experimental results in section 5. Conclusions are given in section 6.

## 2 NOTATION AND TERMINOLOGY

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 $X_1, \ldots, X_k$[1] 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. The decision version of the CSP is determining if a solution exists.

For an instance $\Pi$ of CSP with $n$ variables, its *constraint hypergraph* $G^{\Pi}$ has $n$ vertices $v_1, \ldots, v_n$, which correspond to the variables of $\Pi$ 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 $\Pi$ will be used; the incompatibility hypergraph of $\Pi$, $C^{\Pi}$, is an $n$-partite hypergraph of which the $i$th part corresponds to variable $X_i$ of $\Pi$ which has exactly $|D_i|$ vertices, one for each value in $D_i$. In $C^{\Pi}$ 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[2]. We define the Boolean function $\phi$ on the search space $S$ as the *feasibility condition*, with $\phi(s) = \mathit{true}$ if and only if $s$ is a solution of $S$, while the set $\{s \in S | \phi(s) = \mathit{true}\}$ will be called the *feasible search space*.

Note that for the sake of simplicity we study binary 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 ([17]).

[1]We use shorthand $c_j$ and $\mathcal{C}^j$ if it cannot lead to confusion.

[2]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

## 3 GENERATING CSPS

In [1], Achlioptas et al. show that the so-called Models A to D 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 [15, 16] works 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 [16], 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). They further show that one source of this asymptotic insolubility is the appearance of 'flawed' values, i.e., values that are incompatible with all the values of some other variable. A number of experimental studies, as reported in [14] have avoided this pitfall, but many others did not.

In the same paper, Achlioptas at al., proposed an alternative model for generating random CSP instances (Model E), which 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 ([18]). This model is defined as:

**Definition 1** *$C^{\Pi}$ 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. The paper continues by stating that for a random instance $\Pi$ generated using Model E, if we have $r < 1/2$, $\Pi$ almost certainly has a solution and it is possible to bound the underconstrained and overconstrained regions.

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. In [1] Achlioptas et al. show that Model E also exhibits a phase transition when one of its variables is changed and, they give 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 between the under- and overconstrained regions.

## 4 ADAPTIVE FITNESS FUNCTIONS FOR CSPS

### 4.1 STANDARD STEPWISE ADAPTION OF WEIGHTS

The Stepwise Adaptation of Weights (SAW) mechanism has been introduced by Eiben and van der Hauw [7, 8]. In several comparisons the SAW EA proved to be a superior technique for solving specific CSPs [9, 2]. The basic idea behind the SAW mechanism is that constraints that are not satisfied after a certain number of search steps (i.e. fitness evaluations), must be hard and therefore be given more attention. This is realized by using a weighted sum of constraint violations as fitness function and varying these weights to direct the search. Technically, all weights are given an initial value of 1 and re-setting them happens by adding a value $\Delta w$ after a certain predefined number of evaluations. The best individual of the given population is used as reference for weight updates. Constraints that are violated in the current-best-individual are given a higher weight during an update operation.

The two penalty systems we compare here differ in the elementary penalty terms the fitness function is composed from. Namely, these terms can be based on:

1. *constraints* that are violated, or on
2. *variables* that are wrongly instantiated.

These two mechanisms can formally be described as follows:

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

where

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

respectively

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

where

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

Obviously, for the above functions $f_1, f_2$ and for each $s \in S$ we have that $\phi(s) = \mathit{true}$ if and only if $f_i(s) = 0$ with $i \in \{1, 2\}$.

The corresponding adaption schemes, that is, weight update mechanisms, are as follows:

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

respectively

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

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

### 4.2 REFINING FUNCTIONS

In [12] and [13], Gottlieb and Voss have shown that refining functions improve the performance of SAW for 3-SAT problems. Here we investigate if using these refining functions also improves performance on randomly generated binary CSPs. Extending the SAW EA with a refining function means to add a term $\alpha \cdot r(X)$ to the fitness functions leading to the following definitions:

$$f_3(s) = \sum_{i=1}^{m} w_i \cdot \chi(s, c_i) + \alpha \cdot r(X) \qquad (3)$$

respectively

$$f_4(s) = \sum_{i=1}^{n} w_i \cdot \chi(s, \mathcal{C}^i) + \alpha \cdot r(X) \qquad (4)$$

Adding $\alpha \cdot r(X)$ to the fitness function makes it possible to differentiate between individuals having the same basic fitness value. Note that the refining function values are limited to the range $[0, 1)$. By using a refining factor $\alpha$, the influence of the refining function on the fitness function can be tuned. The original definition of the refining function used in [13] is adapted to CSPs as follows:

$$r(X) = \frac{1}{2}\left(1 + \frac{\sum_{j=1}^{n} K(X_j) \cdot v_j}{1 + \sum_{j=1}^{n} |v_j|}\right)$$

with

$$K(X_j) = \begin{cases} 1 & \text{if } X_j \text{ is not causing a violation in } X^* \\ -1 & \text{otherwise} \end{cases}$$

where weight $v_j$ belongs to variable $X_j$. Note that $r(X)$ always adds a term concerning the wrongly instantiated variables and we get two separate sets of weights. In case of $f_1$ ($f_3$) we get weights $w_i$ with $i \in \{1, \ldots, m\}$ for the constraints and weights $v_j$ with $j \in \{1, \ldots, n\}$ for the variables. In case of $f_2$ ($f_4$) both sets of weights $w_i$ with $i \in \{1, \ldots, n\}$ and $v_j$ with $j \in \{1, \ldots, n\}$ concern the variables.

In both cases the update rule SAW$_{con}$, respectively SAW$_{var}$ can be used for the $w$'s and we introduce a new rule for the $v$'s. The values for the $v$'s are also initiated with 1 and updated simultaneously with the $w$'s.

Following [13] we, in fact, introduce two different update rules for the weights in the refinement function. The rule AW1 is a problem-independent version which reflects a moderate adjustment of the weights towards the complement of the current best individual $X^*$,

$$v_j \leftarrow v_j - K(x_j^*) \quad \text{for } j \in \{1, \ldots, n\} \quad \text{(AW1)}$$

The rule AW2 is a problem-dependent version that uses CSP specific knowledge in $\mathcal{C}^l$:

$$v_j \leftarrow v_j - \sum_{l=1}^{n} K(x_l^*)|\,\mathcal{C}^l(x_l^*)| \quad j \in \{1, \ldots, n\} \quad \text{(AW2)}$$

AW2 takes into account that it is necessary to change a variable that has an unsatisfied constraint that binds it in order to improve the current solution and hence guides the EA towards solutions satisfying yet unsatisfied constraints. We denote the SAW algorithm that uses $f_3$ with update rule $AW1$ as SAW$_{con,ref,AW1}$ and if it uses $f_4$ with update rule $AW1$ we use SAW$_{var,ref,AW1}$. We replace $AW1$ subscript with $AW2$ if the SAW algorithm uses the $AW2$ update rule.

### 4.3 DECAY

In [11], Frank showed that WGSAT, a local search algorithm using clause weights, is susceptible to large absolute weights and convergence of relative weights. To overcome this problem he suggested a decay factor. A decay factor yields a chance to reduce high absolute weights, which allows a correction of inappropriately adapted weights. Given the decay factor $\beta \in [0, 1]$, we consider the decayed adaption schemes:

$$\begin{array}{ll} w_i \leftarrow \beta w_i + \chi(X^*, c_i) & (\text{SAW}_{var,d}) \\ w_i \leftarrow \beta w_i + \chi(X^*, \mathcal{C}^i) & (\text{SAW}_{con,d}) \\ v_j \leftarrow \beta v_j - K(x_j^*) & (\text{AW1}_d) \\ v_j \leftarrow \beta v_j - \sum_{l=1}^{n} K(x_l^*)|\,\mathcal{C}^l(x_l^*)| & (\text{AW2}_d) \end{array}$$

As already observed for WGSAT in [11] and for SAW in [13], for 3-SAT, the good $\beta$-values are very close to 1. The same behavior for large decay rates occurred using the SAW mechanisms for CSPs as it did for 3-SAT. If the decay rate was too large ($\beta$ too small, approximately $\beta \leq 0.9$), it destroyed much of the information learned during the search process.

## 5 EXPERIMENTAL RESULTS

We used a steady-state evolutionary algorithm with order-based representation that proved to be the best option in [10]. Here an individual is a permutation of the variables and a 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. (Technically, it is instantiated to a special value indicating a conflict.) Earlier work has shown that small populations produce the best performance and that for CSPs a population size of just 1 is optimal. Therefore, we use a $(1+1)$ style EA and no crossover operation is needed. The mutation operator is a simple swap operator, which randomly chooses one variable in the given permutation and swaps it with another randomly chosen variable. The initial population is generated randomly and the weights are adapted each time 250 evaluations have been done. (Preliminary experiments with different adaption periods showed little differences in performance; 250 gave just slightly better results than other values.) After a maximum of 100,000 evaluations, the runs were terminated. As mentioned in section 3, we used a set of 10 different probabilities for the Model E CSP generator. With each of these probabilities we generated 25 instances and performed 10 runs over each instance, resulting in 250 runs for every $p$ value for each algorithm variant.

We used two measures of comparison for the algorithms; first Success Rate (SR), which denotes the percentage of the runs that were completed with a solution[3]; second, Average number of Evaluations to a Solution (AES). Note that the last measure is only defined when a solution was found and that, although it seems a 'fair' measure, it could be misleading as some EAs use 'hidden labor' which could be invisible to the measure. An example of hidden labor could be some of the work done in $(AW2)$, where problem-specific knowledge was used in the fitness function.

We experimented with SAW algorithms using the different fitness functions $(f_1, f_2, f_3, f_4)$, using the two refining functions ($AW1$ and $AW2$) for fitness functions $f_3$ and $f_4$ and using the decay mechanisms.

The results are depicted in figure 1, the corresponding numerical figures are given in table 1. These outcomes show that there is little difference between the SAW algorithms that consider either constraints or variables (fitness functions $f_1$ or $f_2$). It might be observed that SAW$_{var}$ has a small advantage in Average Evaluations to Solution (AES) but when comparing Success Rate (SR), the differences are small. In instances with probability 0.34 and 0.38 SAW$_{var}$ has a better SR, while in instances with probability 0.36 SAW$_{con}$ solved more instances. Other instances were solved by both algorithms equally well (SR) and until $p = 0.24$ the speed figures (AES) are also the same. The standard deviations of the averages for AES (not presented here) were also so close that no further distinction could be made.

[3]We use the decimal notation of a percentage: $10\% = 0.10$

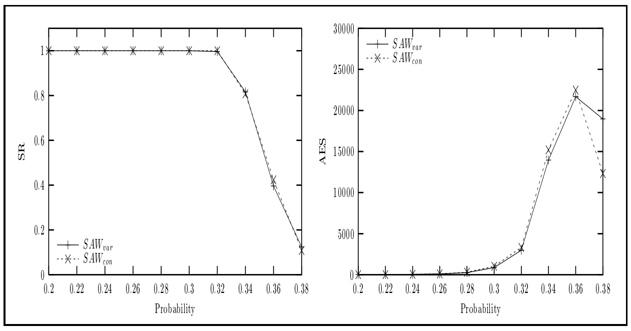

Figure 1: SR and AES graphs for SAW$_{var}$ and SAW$_{con}$

The little difference between trying to solve CSPs with either penalizing variables or constraints is somewhat surprising if we consider that there are much more constraints than variables. This implies that SAW$_{con}$ working with the fitness function $f_1$ has more information, but apparently $f_2$ is already "strong" enough.

| | SAW$_{var}$ | | SAW$_{con}$ | |
|---|---|---|---|---|
| | SR | AES | SR | AES |
| 0.20 | 1 | 9.936 | 1 | 9.936 |
| 0.22 | 1 | 17.304 | 1 | 17.304 |
| 0.24 | 1 | 45.632 | 1 | 45.632 |
| 0.26 | 1 | 104.448 | 1 | 106.336 |
| 0.28 | 1 | 258.28 | 1 | 311.068 |
| 0.30 | 1 | 870.556 | 1 | 1023.66 |
| 0.32 | 1 | 2984.63 | 1 | 3322.18 |
| 0.34 | 0.816 | 13962.6 | 0.808 | 15212.3 |
| 0.36 | 0.396 | 21683.3 | 0.424 | 22480.3 |
| 0.38 | 0.124 | 18968.3 | 0.108 | 12332.2 |

Table 1: Numerical results for SAW$_{var}$ and SAW$_{con}$

The results of the experiments with the extended SAW mechanism are given in table 2 for SAW$_{var,ref,AW1}$ and SAW$_{con,ref,AW1}$ and in table 3 for SAW$_{var,ref,AW2}$ and SAW$_{con,ref,AW2}$. The figures show that the addition of the refining function AW1 produced no improvement at all. Experiments were also performed with varying refinement factors, even up to values where the performance of the algorithms began to deteriorate. Differences between the refining functions (AW1 and AW2) are also small, which indicates that adding extra domain information, in the form of $\mathcal{C}^l$, also did not improve search performance. This is also surprising because one would expect to increase search speed when incorporating extra domain knowledge. This might come at the cost of premature convergence because of searching too greedily

| | SAW$_{var,ref,AW1}$ | | SAW$_{con,ref,AW1}$ | |
|---|---|---|---|---|
| | SR | AES | SR | AES |
| 0.20 | 1 | 9.936 | 1 | 9.936 |
| 0.22 | 1 | 17.304 | 1 | 17.304 |
| 0.24 | 1 | 45.632 | 1 | 45.632 |
| 0.26 | 1 | 104.448 | 1 | 106.336 |
| 0.28 | 1 | 258.28 | 1 | 311.068 |
| 0.30 | 1 | 870.556 | 1 | 1023.66 |
| 0.32 | 1 | 2984.63 | 1 | 3322.18 |
| 0.34 | 0.816 | 13962.6 | 0.808 | 15212.3 |
| 0.36 | 0.396 | 21683.3 | 0.424 | 22480.3 |
| 0.38 | 0.124 | 18968.3 | 0.108 | 12332.3 |

Table 2: Results for SAW$_{var,ref,AW1}$ and SAW$_{con,ref,AW1}$

| | SAW$_{var,ref,AW2}$ | | SAW$_{con,ref,AW2}$ | |
|---|---|---|---|---|
| | SR | AES | SR | AES |
| 0.20 | 1 | 9.936 | 1 | 9.936 |
| 0.22 | 1 | 17.304 | 1 | 17.304 |
| 0.24 | 1 | 45.632 | 1 | 45.632 |
| 0.26 | 1 | 104.448 | 1 | 104.524 |
| 0.28 | 1 | 258.28 | 1 | 252.996 |
| 0.30 | 1 | 870.556 | 1 | 1048.48 |
| 0.32 | 0.996 | 2984.63 | 1 | 3430.64 |
| 0.34 | 0.816 | 13962.6 | 0.840 | 16487.5 |
| 0.36 | 0.396 | 21683.3 | 0.368 | 20957.4 |
| 0.38 | 0.124 | 18968.3 | 0.116 | 23669.4 |

Table 3: Results for SAW$_{var,ref,AW2}$ and SAW$_{con,ref,AW2}$

When interpreting these results, recall that the refining function was designed to distinguish between individuals having the same basic fitness value. The lack of improvement when using refining functions seems to imply that in case of binary CSPs, the basic functions $f_1$ and $f_2$ contain sufficient information to guide the search successfully.

Experiments using decay factor for solving SAT problems showed that values around $\beta = 1$ work best, cf. [11, 13]. Our studies with random binary CSPs also indicated the same. However, we also found that applying decay to the SAW technique did not change algorithm performance significantly when $\beta$ values were taken from the set $\{0.95, 0.96, 0.97, 0.98, 0.99, 1.00\}$. Note that $\beta = 1.00$ amounts to no decay. Therefore this observation implies that algorithm variants without decay and with decay using a good $\beta$ value are not performing differently.

Figure 2 gives an illustration by showing typical runs using different decay factors. These runs were performed on instances generated with probability 0.32. Note that the figure only presents the search speed results (AES). The success rates are not given because all algorithms solved all instances, except for SAW$_{var,ref,AW1,d}$ with decay factor 0.98 which solved 096% of the instances (a SR of = 0.96). The curves indicate a difference between SAW$_{var,ref}$ and the other algorithms. Namely, SAW$_{var,ref}$ using either AW1 or AW2 seems to improve when switching off decay, i.e. for $\beta = 1.0$. For all other algorithms a decay factor has no significant influence on the search speed. Recall that the decay of weights in the SAW fitness function was added to actively suppress the growth of weight values. These results indicate that such a growth – identified as dangerous in related fields, cf. [11, 13], – either does not occur or is not harmful in case of random binary CSPs.

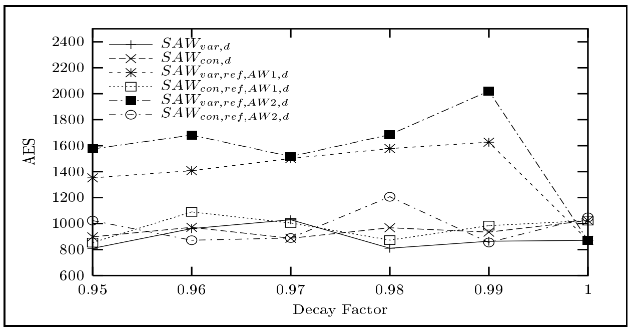

Figure 2: graphs of SAW$_{var,d}$ and SAW$_{con,d}$ with different decay factors. $p = 0.32$, $\Delta = 250$, refinement-factor $\alpha = 40$

## 6 CONCLUSIONS

The research presented in this paper had a twofold objective: presenting and illustrating a recently proposed problem instance generator for binary CSPs and comparing a number of variants and extensions of the SAW technique. In particular, we used a CSP generator based on the so-called model E in [1]. We found that the phase transition effects were clearly observable and by the use of one single hardness parameter the results were easier to present – and to interpret – than in case of the formerly used two-parameter-based generators [3, 4, 5, 10]. The new generator deserves a recommendation for further experimental research.

As for the algorithm variants, we experimented with the unextended version of the SAW technique with two penalty systems: one calculated over the variables and one calculated over the constraints. They turned out to yield very similar performance, which is surprising. The constraint based fitness function, after all, is based on more information as there are usually more constraints than variables and one would expect to improve a search algorithm when using more information.

Inspired by related work on satisfiability problems we have also experimented with two extensions of SAWing. In particular, we tried refining functions and a decay mechanism. A refining function $\alpha \cdot r(X)$ added to the fitness function makes it possible to differentiate between individuals having the same basic fitness value. We found that the application of refining functions did not improve performance and that varying the refining factor $\alpha$ did not have any influence on performance. These results seem to imply that in case of binary CSPs the basic functions $f_1$ and $f_2$ contain sufficient information to guide the search successfully.

The addition of the decay factor did not improve the performance in general either, neither for the original SAW techniques, nor for the SAW technique with a refining function. In fact it had a negative effect on SAW$_{var,ref}$. This might be the consequence of the relatively small problem size, where the accumulation of large relative weights stays within limits and thus does not need a counterforce. This finding points to the same direction as our conclusion about refinement functions: the basic SAW mechanism is powerful enough to solve random binary CSPs.

All in all, the comparison of the algorithm variants shows a surprising, but pleasant picture: The simplest setup (SAW with variable related penalties, no extensions) is as good or better than any of the more sophisticated variants. Although current and future research will undoubtfully refine this picture, for the time being this is good news for algorithm designers.

Further research is carried out with new types of refining functions, problems with varying sizes (scale-up), and larger population sizes.

## 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] Th. Bäck, A.E. Eiben, and M.E. Vink. A superior evolutionary algorithm for 3-SAT. In V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, editors, *Proceedings of the 7th Annual Conference on Evolutionary Programming*, number 1477 in Lecture Notes in Computer Science, pages 125–136, Berlin, 1998. Springer-Verlag.

[3] J. Bowen and G. Dozier. Solving constraint satisfaction problems using a genetic/systematic search hybride that realizes when to quit. In L.J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 122–129. Morgan Kaufmann Publishers, Inc., 1995.

[4] G. Dozier, J. Bowen, and D. Bahler. Solving small and large constraint satisfaction problems using a heuristic-based microgenetic algorithm. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation*, pages 306–311. IEEE Computer Society Press, 1994.

[5] G. Dozier, J. Bowen, and A. Homaifar. Solving constraint satisfaction problems using hybrid evolutionary search. *Transactions on Evolutionary Computation*, 2(1):23–33, 1998.

[6] A.E. Eiben, Th. Bäck, M. Schoenauer, and H.-P. Schwefel, editors. *Proceedings of the 5th Conference on Parallel Problem Solving from Nature*, number 1498 in Lecture Notes in Computer Science, Berlin, 1998. Springer-Verlag.

[7] A.E. Eiben and J.K. van der Hauw. Solving 3-SAT with adaptive Genetic Algorithms. In *Proceedings of the 4th IEEE Conference on Evolutionary Computation*, pages 81–86. IEEE Computer Society Press, 1997.

[8] 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 Lecture Notes in Computer Science, pages 95–106, Berlin, 1998. Springer-Verlag.

[9] A.E. Eiben, J.K. van der Hauw, and J.I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

[10] A.E. Eiben, J.I. van Hemert, E. Marchiori, and A.G. Steenbeek. Solving binary constraint satisfaction problems using evolutionary algorithms with an adaptive fitness function. In Eiben et al. [6], pages 196–205.

[11] J. Frank. Learning short-term weights for GSAT. Technical report, University of California at Davis, oct 1996.

[12] J. Gottlieb and N. Voss. Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. In Eiben et al. [6].

[13] J. Gottlieb and N. Voss. Adaptive fitness functions for the satisfiability problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel, editors, *Proceedings of the 6th Conference on Parallel Problem Solving from Nature*, number 1917 in Lecture Notes in Computer Science, Berlin, 2000. Springer-Verlag.

[14] E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh. Random constraint satisfaction: theory meets practice. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 325–339, Berlin, 1998. Springer-Verlag.

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

[16] 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.

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

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