Evolutionary Algorithms for the Satisfiability Problem

Jens Gottlieb jens.gottlieb@sap.com
SAP AG, Neurottstrasse 16, 69190 Walldorf, Germany

Elena Marchiori elena@cs.vu.nl
Department of Computer Science, Free University Amsterdam, de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

Claudio Rossi rossi@dsi.unive.it
Department of Computer Science, Ca' Foscari University of Venice, Via Torino 155, 31072 Mestre, Italy

Abstract

Several evolutionary algorithms have been proposed for the satisfiability problem. We review the solution representations suggested in literature and choose the most promising one – the bit string representation – for further evaluation. An empirical comparison on commonly used benchmarks is presented for the most successful evolutionary algorithms and for WSAT, a prominent local search algorithm for the satisfiability problem. The key features of successful evolutionary algorithms are identified, thereby providing useful methodological guidelines for designing new heuristics. Our results indicate that evolutionary algorithms are competitive to WSAT.

Keywords

Satisfiability problem, evolutionary algorithm, local search, adaptive fitness function, WSAT.

1 Introduction

The satisfiability problem (SAT) is a paradigmatic NP-complete problem (Cook, 1971) with many relevant practical applications (Du et al., 1997), like Boolean circuit synthesis (Brayton et al., 1985) and test pattern generation (Larrabee, 1992). SAT is based on a set of Boolean variables $x_1, \ldots, x_n$ and a Boolean formula $f : \mathbb{B}^n \to \mathbb{B}$, $\mathbb{B} = \{0, 1\}$, and the question is whether a variable assignment $x = (x_1, \ldots, x_n) \in \mathbb{B}^n$ exists such that $f(x) = 1$. A SAT instance is called *satisfiable* if such x exists, and *unsatisfiable* otherwise. The formula f is in conjunctive normal form if $f(x) = c_1(x) \wedge \cdots \wedge c_m(x)$, where each clause c_i is a disjunction of literals, and a literal is a variable or its negation. SAT instances can be assumed having conjunctive normal form without loss of generality (Tseitin, 1968), and the class k-SAT contains those with each clause containing exactly k distinct literals. While 2-SAT is solvable in polynomial time, k-SAT is NP-complete for $k \geq 3$ (Garey and Johnson, 1979).

Due to their theoretical and practical relevance, SAT and, in particular, 3-SAT have been studied extensively, and many exact and heuristic algorithms have been introduced (Battiti and Protasi, 1998; Du et al., 1997). Exact algorithms give a definite answer to any problem instance (be it satisfiable or unsatisfiable) but have an exponential

worst-case complexity, unless P = NP. Heuristic algorithms can find solutions to satisfiable instances quickly, but they are not guaranteed to give a definite answer to all problem instances.

Evolutionary algorithms (EAs) are heuristic algorithms that have been applied to SAT and many other NP-complete problems. Some negative results question the basic ability of EAs to solve SAT. De Jong and Spears (1989) proposed a classical genetic algorithm (GA) for SAT and observed that the GA may not outperform highly tuned, problem-specific algorithms. This result was confirmed experimentally by Fleurent and Ferland (1996), who reported scarce performance of pure GAs when compared to local search. Furthermore, Rana and Whitley (1998) showed classical GAs being unsuitable for the MAXSAT fitness function, which counts the number of satisfied clauses, because the corresponding domain contains misleading low-order schema information, and the search space tends to result in similar schema fitness averages. Recent results showed that EAs can nevertheless yield good results for SAT if equipped with additional techniques to overcome the weaknesses of classical GAs. These techniques include adaptive fitness functions, problem-specific variation operators, and local optimization (Bäck et al., 1998; Eiben and van der Hauw, 1997; Fleurent and Ferland, 1996; Folino et al., 1998; Gottlieb and Voss, 1998a, 2000; Marchiori and Rossi, 1999; Rossi et al., 2000).

This paper reviews EAs for SAT, focusing on alternative solution representations and different techniques to enhance the performance of classical GAs. The most promising evolutionary algorithms, which employ the bit string representation, are examined on three commonly used benchmark suites. The results identify the main features of successful EAs for SAT, and they also demonstrate that EAs compare favorably to WSAT, a popular local search heuristic for SAT.

The paper is organized as follows. Section 2 introduces solution representations that have been proposed for SAT, and Section 3 describes selected EAs and WSAT. Performance evaluation is discussed in Section 4. An experimental comparison on benchmarks is presented in Section 5, which also includes a discussion of the relation to other heuristics. Section 6 draws conclusions and addresses future research directions.

2 The Representation Issue

2.1 The Bit String Representation

The most obvious way to represent a solution candidate for SAT is a bit string of length n, where every variable is associated to one bit. While this representation is self-explanatory, the choice of an appropriate fitness function is important, which can be observed by the great variety of distinct functions proposed in literature (De Jong and Spears, 1989; Eiben and van der Hauw, 1997; Frank, 1994; Gottlieb and Voss, 1998a). The original Boolean function f itself might be used as fitness function since SAT solutions correspond to global optima of f. However, this approach typically fails because the EA degenerates to pure random search as all solution candidates have fitness 0 unless a solution is found. Except for the fitness function proposed by De Jong and Spears (1989) for arbitrary SAT instances, all other fitness functions assume conjunctive normal form. In the MAXSAT formulation, the fitness value is equivalent to the number of satisfied clauses, i.e.,

$$f_{MAXSAT}(x) = c_1(x) + \ldots + c_m(x),$$

where $c_i(x)$ represents the truth value of the ith clause. This fitness function is used in most EAs for SAT (Fleurent and Ferland, 1996; Folino et al., 1998; Frank, 1994; Marchiori and Rossi, 1999; Park, 1995; Rossi et al., 2000). As EAs based on this fitness function

have difficulty solving even small SAT instances when classical variation operators are used, alternative functions were suggested that rely on adaptation mechanisms to allow a fine-grained distinction of solution candidates (Eiben and van der Hauw, 1997; Gottlieb and Voss, 1998a). Many different variation operators – ranging from standard bit mutation and 2-point crossover (De Jong and Spears, 1989) to problem-specific operators (Gottlieb and Voss, 1998a) and local optimization (Marchiori and Rossi, 1999) – have been used within bit string based EAs. The most successful EAs for SAT use this representation, together with non-standard variation operators or adaptive fitness functions; Section 3 describes these approaches in greater detail.

2.2 The Floating Point Representation

The floating point representation was suggested by Bäck et al. (1998), who proposed to transform SAT into a continuous optimization problem, which then can be tackled by classical evolutionary techniques for numerical optimization. Solution candidates are represented by continuous vectors $y \in [-1,1]^n$, and the fitness function is designed such that global optima directly correspond to feasible solutions for SAT. The transformation is based on replacing literals x_j and $\overline{x_j}$ by $(y_j-1)^2$ and $(y_j+1)^2$ and substituting the Boolean operators $\wedge$ and $\vee$ by their arithmetical counterparts $\cdot$ and $+$, respectively. To illustrate this approach, we consider the 3-SAT instance given by the formula

$$f(x) = (x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_1} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_4) \ . \tag{1}$$

The function $f : \mathbb{B}^4 \to \mathbb{B}$ is transformed into the continuous function $g : [-1,1]^4 \to \mathbb{R}$ according to

$$g(y) = (y_1-1)^2(y_2+1)^2(y_4-1)^2 + (y_1+1)^2(y_3-1)^2(y_4+1)^2 + (y_2+1)^2(y_3+1)^2(y_4-1)^2,$$

which is to be minimized. Note that the Boolean values 0 and 1 are associated with -1 and 1. In their implementation, Bäck et al. converted continuous vectors by rounding negative and positive values to -1 and 1 in order to check whether a solution for SAT already is represented.

Although the experiments with standard variation operators used in evolution strategies showed the basic ability of the floating point representation to find solutions for SAT, the overall performance is clearly inferior to SAWEA, which uses the bit string representation and an adaptive fitness function (Bäck et al., 1998; Eiben and van der Hauw, 1997), and which we present in Section 3.1.

2.3 The Clausal Representation

Hao (1995) proposed the clausal representation that emphasizes the local effects of the variables in the clauses. This representation selects feasible variable assignments for each clause, and it aims at finding globally consistent partial variable assignments. Recalling the Boolean function f from Equation (1), eight variable assignments exist for the variables occurring in the first clause, and only one of them causes a clause violation: $(x_1, x_2, x_4) = (0,1,0)$. In the clausal representation, a solution candidate is a combination of m variable assignments, where each defines all variables contained in one clause and only feasible variable assignments are allowed. The solution candidate

$$((x_1, x_2, x_4) = (1,0,0), (x_1, x_3, x_4) = (1,1,0), (x_2, x_3, x_4) = (0,1,1))$$

combines feasible assignments for the clauses but also contains inconsistent assignments for x_4. If a solution candidate was globally consistent, a (complete) variable assignment could be derived directly. Therefore, an EA using the clausal representation

must be guided by a fitness function reflecting the global amount of inconsistencies among the variable assignments for the clauses. Hao (1995) made several proposals, including fitness functions that count the number of inconsistencies and weighted variants of them. In contrast to the previously described variable-oriented approaches, this representation implies a search process that focuses on the relations between variables that are linked together by distinct clauses.

The clausal representation allows the use of classical variation operators as solutions are also represented by bit strings; nevertheless care must be taken to ensure local consistency. Hao (1995) argued the elimination of unfeasible variable assignments yields a reduction of the search space. However, as $m \cdot k$ bits are needed to represent a solution candidate for k-SAT, the representation induces much larger search spaces for instances with many clauses when compared to the bit string representation. We are not aware of extensive computational studies for this representation, but we expect a similar performance as for the path representation due to the similarity of both representations' underlying ideas.

2.4 The Path Representation

The path representation was suggested by Gottlieb and Voss (1998b) and exploits the fact that a feasible solution must satisfy at least one literal in each clause. In order to determine a solution, we may select one literal in each clause, yielding a path through all clauses that visits each clause exactly once. If there is no inconsistency between the literals in such a path, a complete variable assignment can be constructed immediately. Given the SAT instance (1), the path $(x_1, \overline{x_4}, \overline{x_3})$ is feasible and induces the complete variable assignments $x = (1, 0, 0, 0)$ and $x' = (1, 1, 0, 0)$, while the path $(x_1, \overline{x_4}, x_4)$ contains an inconsistency. A reasonable fitness function for this representation must measure the amount of inconsistencies. While this corresponds to the clausal representation, one relevant difference between both codings is the length of solution candidates. As the choice of one literal in a clause can be encoded by $\log_2 k$ bits, the total number of bits needed is $m \cdot \log_2 k$, which results in a more compact representation. Another distinguishing feature is that the clausal representation aims at determining exactly one feasible solution, while a path is able to represent a family of feasible solutions.

The path representation was evaluated in an experimental study (Gottlieb and Voss, 1998b) relying on problem-specific variation operators and fitness functions based on the number of inconsistencies among the selected literals. As the obtained results were discouraging compared to the bit string representation, no further attempts have been made to improve this representation.

3 Selected Evolutionary Algorithms and WSAT

Effective EAs for SAT using bit string representation differ from each other on some of the following main features: replacement scheme, parent selection, mutation, crossover, and fitness function. Moreover, some EAs incorporate non-genetic features like adaptation and local search. Table 1 summarizes the features of the EAs mainly reviewed in this paper; note that they all rely on pure random initial populations. While the approaches SAWEA (Bäck et al., 1998; Eiben and van der Hauw, 1997) and RFEA (Gottlieb and Voss, 1998a; Gottlieb and Voss, 2000) are based on adaptive fitness functions, the other two EAs, FlipGA (Marchiori and Rossi, 1999) and ASAP (Rossi et al., 2000), use the MAXSAT fitness function and incorporate local search.

Table 1: Main features of selected evolutionary algorithms.

Feature	SAWEA	RFEA	FlipGA	ASAP
replacement	$(1, \lambda^*)$	steady-state	generational	$(1+1)$
parent selection	–	tournament	fitness proportional	–
fitness	f_{SAW}	f_{REF}	f_{MAXSAT}	f_{MAXSAT}
initialization	random	random	random	random
crossover	–	–	uniform	–
mutation	MutOne	knowledge-based	random	random adaptive
local search	–	–	flip heuristic	flip heuristic
adaptation	fitness	fitness	–	tabu list

In addition to the EAs, we consider WSAT (Selman et al., 1994) for comparison purposes. WSAT is a popular local search heuristic that evaluates solutions by the number of satisfied clauses.

3.1 SAWEA: Using Stepwise Adaptation of Weights

Eiben and van der Hauw (1997) examined evolutionary algorithms for 3-SAT, which use the *stepwise adaptation of weights* (SAW) principle and the fitness function

$$f_{SAW}(x) = w_1 \cdot c_1(x) + \ldots + w_m \cdot c_m(x) \, .$$

The weights $w_i \in \mathbb{N}$ are adapted in order to identify those clauses that are difficult to satisfy in the current search phase. In the beginning, all weights are initialized by $w_i = 1$, which means that the MAXSAT fitness function is used. After certain time periods of 250 fitness evaluations, the weights are adjusted according to $w_i \leftarrow w_i + 1 - c_i(x^*)$, where x^* is the current fittest individual. This adaptation scheme increases only those weights that correspond to unsatisfied clauses in x^*, and therefore the weights reflect the "hardness" of the associated clauses. This implicitly forces the evolutionary search to focus on these "difficult" clauses, and hence the search process is guided by the weights.

The SAW principle was further studied by Bäck et al. (1998), who identified the most promising configuration as follows: They employed the MutOne operator, which flips exactly one uniformly chosen bit, and an extinctive $(1, \lambda^*)$ replacement scheme, where the parameter λ^* was fine-tuned for each problem size n. That evolutionary algorithm, referred to as SAWEA, was further improved by de Jong and Kosters (1998), who suggested applying an additional operator to one of the offsprings produced by MutOne. This additional operator randomly selects some clauses and flips one uniformly chosen variable in each selected clause that is not satisfied yet. This approach is called *Lamarckian structural error assignment* and has outperformed SAW on a large, 3-SAT benchmark suite; we refer to it as LSAWEA.

3.2 RFEA: Using Refining Functions

The original idea of *refining functions* was inspired by the observation that many different bit strings exist having the same quality concerning the MAXSAT fitness formulation, which makes it impossible for an EA to distinguish between them (Gottlieb and

Voss, 1998b). Additional heuristic knowledge can be captured in a refining function $r : \mathbb{B}^n \to [0, 1)$ and used within the refined fitness function

$$f_{REF}(x) = c_1(x) + \ldots + c_m(x) + \alpha \cdot r(x) \ ,$$

where the influence of r is controlled by $\alpha > 0$. Using an influence level $\alpha \in [0, 1)$ allows discrimination between chromosomes satisfying the same number of clauses. Higher influence levels bias the evolutionary search towards r, which can be helpful if r is designed and adapted in order to escape from local optima. Gottlieb and Voss (1998a) introduced different kinds of refining functions and investigated the influence of the parameter α. Although adaptive parameter control of α is promising, the results presented in Section 5 are based on using a constant value for α, which is determined by some a priori experiments. Recently, Gottlieb and Voss (2000) presented improvements that led to the best performing EA based on a refining function, which is called RFEA and is described in the following.

RFEA uses population size 4, parent selection by tournaments of size 2, and a steady-state replacement scheme based on eliminating the worst individual. Duplicate elimination is used, i.e., the offspring is rejected if it is already contained in the current population. The only variation operator is a mutation operator that selects an unsatisfied clause and flips exactly one randomly chosen variable contained in the clause.[1] Besides this problem-specific mutation operator, the major component of the EA is the refining function

$$r(x) = \frac{1}{2}\left(1 + \frac{\sum_{j=1}^{n} K(x_j)v_j}{1 + \sum_{j=1}^{n} |v_j|}\right) ,$$

where $K : \mathbb{B} \to \{-1, 1\}$ is defined by $K(1) = 1$ and $K(0) = -1$, and $v_j \in \mathbb{R}$ is the weight associated to the variable x_j. High positive weights indicate that the corresponding variables are favored to be 1, while negative weights express a preference to 0. Initially, the weights are set to 0 and then adapted such that strong preferences are represented by high (absolute value) weights.

The adaptation of v_j aims at escaping from local optima and is defined by $v_j \leftarrow v_j - K(x^*_j) \cdot |U_j(x^*)|$, where x^* is the current best individual, $U_j(x^*)$ represents the set of unsatisfied clauses containing the corresponding variable, and $|U_j(x^*)|$ denotes its cardinality. This scheme adjusts the weights towards the complement of the current best individual and is referred to as AW2. In addition, we consider AW2+, a hybridized scheme that incorporates SAW to accelerate the adaptation process. The scheme AW2+ applies $v_j \leftarrow v_j - K(x^*_j) \sum_{i \in U_j(x^*)} w_i$, where the clause weights w_i are adapted according to SAW. Note that AW2 is a special case of AW2+ with $w_i = 1$. We refer the reader to (Gottlieb and Voss, 2000) for more details on these adaptation mechanisms and use RFEA2 and RFEA2+ to refer to the refining function based EAs employing the adaptation schemes AW2 and AW2+, respectively.

3.3 FlipGA: Using the Flip Heuristic

Marchiori and Rossi (1999) introduced FlipGA, an evolutionary local search algorithm that generates offspring by standard (blind) genetic operators and subsequent improvement by means of local search. FlipGA employs population size 10, fitness proportional parent selection, and a generational replacement scheme with elitism, copying the best

[1] Note the equivalence to the additional operator used by de Jong and Kosters (1998) if restricted to one unsatisfied clause only.

two individuals of the current population into the population of the next generation (De Jong, 1975). Uniform crossover (Syswerda, 1989) is always applied, and a highly disruptive mutation operator is used with probability 0.9. This operator flips each gene with probability 0.5.

The core of FlipGA is the flip heuristic applied to each individual after performing crossover and mutation. The heuristic scans the genes in random order: each gene is flipped, and the flip is accepted if the *gain* (that is, the number of clauses that are satisfied after the flip minus the number of clauses that are satisfied before the flip) is greater than or equal to zero. When all the genes have been considered, the process is repeated if the fitness of the obtained chromosome has been increased with respect to the previous scan of the genes.

The idea behind this approach is to accomplish exploitation and exploration by means of two separate modules: local search and genetic operators, respectively. In this way, one can better control the effect of the different modules and easily modify them for experimental investigation.

3.4 ASAP: Using the Flip Heuristic and Adaptation

The variant of FlipGA introduced by Rossi et al. (2000) is called *adaptive evolutionary algorithm for the satisfiability problem* (ASAP). It is obtained from FlipGA by considering only 1 chromosome, $(1+1)$ replacement that corresponds to classical local optimization, and an adaptive mechanism to control diversification in the search path.

ASAP acts on the chromosome as follows. First, the mutation operator is always applied and flips, for each $j \in [1..n]$, the value of the jth gene with probability $\mu_j \in [0, 0.5]$, where μ_j is adapted during the execution. Next, the resulting chromosome is improved by the flip heuristic like in FlipGA. Moreover, an adaptive mechanism based on tabu search is employed for prohibiting the flip of some variables and for controlling the mutation rate μ_j.

This is realized by means of a table of fixed capacity that is dynamically filled with chromosomes having best fitness. When the best fitness increases, then the table is emptied. When the table is full, the chromosomes in the table are compared gene-wise: those genes that do not have the same value in all the chromosomes are labeled as "frozen." Frozen genes become clamped, that is they are not allowed to be flipped either by mutation or by the flip heuristic. Furthermore, the mutation rate μ_j of the jth gene is set to $F/(2n)$, where F is the number of frozen genes. Finally, the table is also used for restarting the search, which happens when the chromosomes in the (full) table are too similar, where similarity is measured by means of the Hamming distance.

The rationale behind this adaptive mechanism can be explained as follows. The table is filled when the search path has found "capacity" times best chromosomes, which represent local optima with equal fitness. In order to try to escape from the attraction basins of these local optima, only those genes having the same value in all these chromosomes will be allowed to be flipped. The mutation rate is chosen in such a way that the lower the number of not frozen genes is, the higher the probability will be to flip them. The term $1/2$ is used to keep the mutation rate into reasonable limits, since a too-high mutation rate would lose too much information about the search so far, resulting in an almost random jump.

3.5 WSAT: A Local Search Heuristic

Local search heuristics for SAT explore the search space of truth assignments in order to find a solution that maximizes the number of satisfied clauses. Starting from an initial

assignment, which is typically selected at random, local search proceeds by moving from one assignment to another by flipping the truth value of one single variable. The selection of the variable to flip is the crucial phase of the method, and heuristics are used that may include randomness, greediness, and memory.

One of the most popular local search methods for SAT is WSAT (Selman et al., 1994; McAllester et al., 1997). In WSAT, the selection of a variable to be flipped is accomplished in two steps. First, a clause c among those that are currently unsatisfied is randomly selected. Next, a variable appearing in c is selected using a heuristic. McAllester et al. (1997) proposed six heuristics for this selection task. In our experiments, we use the heuristic BEST: with probability P, a random variable appearing in c is selected, otherwise the variable that breaks the fewest number of other clauses when flipped is chosen. The selected variable is flipped to obtain a new assignment. Recall that a variable x breaks a clause if the clause becomes unsatisfied if x was flipped.

The probability P is called *noise*, since, in general, it causes WSAT to perform non-optimal moves, in the sense that they decrease or fail to increase the number of satisfied clauses. Noise helps a local search procedure escape from local optima. In our experiments, we use $P = 0.5$. We refer to Hoos (1998) and Singer et al. (2000) for a thorough analysis of WSAT and related stochastic local search algorithms for SAT.

Observe that WSAT and FlipGA, and hence ASAP, adopt different search strategies. An iteration of WSAT acts locally on one unsatisfied clause and tries to repair it without affecting clauses that are already satisfied. An iteration of FlipGA acts globally on the problem instance and tries to decrease the total number of unsatisfied clauses.

4 Phase Transition, Benchmarks, and Performance Measures

4.1 Phase Transition and Benchmark Instances

Random 3-SAT instances have been used in many studies on the cost of the search for exact and heuristic algorithms, since they represent the challenges faced by an algorithm in the absence of any assumption about the problem domain. In random 3-SAT, the number n of variables is fixed, and the control parameter is m/n (m is the number of clauses). Varying m/n produces a sharp threshold or phase transition in the probability of satisfiability and an associated cost peak for a range of complete algorithms (a phenomenon discussed elsewhere (Mitchell et al., 1992; Larrabee and Tsuji, 1992)). There is a critical value of the control parameter (about 4.3) such that instances generated with the parameter in the region lower than the critical value (the underconstrained region) almost always have solutions. Those generated from the overconstrained region, where the control parameter is higher than the critical value, almost always have no solutions. Recent works have identified properties of hard random k-SAT instances that are related to their backbone, that is, the set of literals logically entailed by an instance. The reader is referred to Singer et al. (2000) for a recent contribution on this subject.

In our experiments, we use random 3-SAT benchmark instances with $m/n = 4.3$ generated using the `mkcnf` generator[2] using the `forced` option to ensure that they are satisfiable. Table 2 presents an overview of these instances, which are grouped into three test suites that are available online[3].

[2] `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances`

[3] `http://www.in.tu-clausthal.de/~gottlieb/benchmarks/3sat`

Table 2: Overview of test suites.

Suite	Total of instances	Instances for each n	Problem size n	First reference
A	12	3	$30, 40, 50, 100$	(Bäck et al., 1998)
B	150	50	50	(Gottlieb and Voss, 1998a)
			$75, 100$	(Marchiori and Rossi, 1999)
C	500	100	$20, 40, 60, 80, 100$	(de Jong and Kosters, 1998)

4.2 Measuring the Quality

In order to obtain statistically significant results, several runs are required for each benchmark instance under consideration. The achieved quality is measured by the *success rate* (SR), which represents the portion of runs where a solution has been found. While its use is undisputed, the SR strongly depends on the computational effort allowed by the termination criterion. Hence, care must be taken when comparing the SR of structurally different EAs, which usually apply different termination conditions.

Another quality measure, mainly used for problems for which the global optimum is unknown, takes into account all the outcomes of the algorithm. In this case, information is provided about the best result obtained in each run, as well as the average and standard deviation of the best results over all the runs. For the satisfiability problem, this amounts to considering the MAXSAT fitness function of the best chromosome as a result of a run (the satisfiability problem is then called MAXSAT instead of SAT).

4.3 Measuring the Computational Cost

In contrast to the quality issue where SR is dominant, different measures have been proposed to quantify the computational cost of an EA. The computational cost of simple genetic algorithms can be estimated using the *average number of evaluations to solution* (AES), which considers the average number of fitness evaluations needed to find a solution in successful runs. This measure does not depend on the machine used and on the actual implementation. However, the applicability of AES is severely limited in a comparison of EAs relying on structurally different fitness functions or variation operators.

Marchiori and Rossi (1999) quantified the cost of FlipGA and ASAP by means of the *average flip cost in terms of number of fitness evaluations to solution* (AFES), which also takes into account the hidden cost of the flip heuristic. For random 3-SAT instances, it was observed by Spears (1996) that the number of clauses that have to be processed for computing the gain of one flip is (on the average) $3m/n$. As m clauses must be processed in a complete fitness evaluation, one complete evaluation corresponds to $n/3$ partial evaluations caused by single flips. Based on this rationale, the average costs of local search that needs F single flips to find a solution can be related to $3F/n$ complete fitness evaluations, which yields the AFES index and thus allows a direct comparison to the AES of algorithms relying only on complete fitness evaluations.

Instead of relating the computational costs to complete fitness evaluations, the efforts can also be measured by the number of the basic moves in the search space, i.e., single bit flips, needed to find a solution. This has become the standard measure used for studying the cost of SAT algorithms (Hoos, 1998; Singer et al., 2000). Gottlieb and Voss (2000) used the *average number of flips to solution* (AFS) to compare EAs generat-

ing new solution candidates by single bit flips. When local search is perceived as a sequence of one-flip mutations, AFS allows a direct comparison of evolutionary local search to other EAs that use mutation operators flipping exactly one bit. Note that AFS is equivalent to AES in this case, and that AFS can be interpreted as the number of partial fitness evaluations needed to find a solution. Comparisons of local search algorithms to hybrid evolutionary algorithms (Folino et al., 1998; Rossi et al., 2000) were based on the same idea as they consider the number of accepted flips and the total number of flips.

The three measures (AES, AFES, and AFS) quantify important parts of the computational costs, but none of them captures the whole cost of an EA involving fitness evaluation, variation operators, and additional mechanisms like adaptation of parameters. We will use AFS in our empirical analysis, since it permits a direct comparison with local search algorithms like WSAT.

We conclude by mentioning the *average running time* (and standard deviation) over successful runs, an efficiency measure that is (almost) never used in EAs but is often used in local search. This index is machine and implementation dependent, so comparative tables must be interpreted carefully – for instance, by estimating execution times of other machines according to the performances reported by Dongarra (1993). Further, this approach requires the use of optimal data structures for each algorithm under consideration, which might be too time-consuming, in particular, when the potential of an algorithm prototype is to be evaluated. In this case, standard timing routines can be applied to the problem at hand, as done in the experiments of the DIMACS Challenge on Satisfiability (Trick and Johnson, 1996).

5 Evaluation of Evolutionary Algorithms for SAT

5.1 Empirical Results

We compare the algorithms presented in Section 3 on all three benchmark suites that were introduced in Section 4.1. Most results are based on new experiments. However, as some results – like those for SAWEA and LSAWEA – are taken from literature, the number of runs per algorithm differs for some test instances. The results for WSAT are based on 10 runs for each instance. For the EAs, the results for suite A are based on 50 independent runs for each instance. The same settings are used for suite B, except for ASAP, which is run 10 times on each instance. On instances of suite C, SAWEA and LSAWEA are run 3 times, RFEA2 and RFEA2+ 4 times, and FlipGA and ASAP 5 times. All algorithms are terminated if a solution is found or the limit of $T = 300\,000$ bit flips is reached.

Tables 3, 4, and 5 present the results for the suites A, B, and C, respectively. In general, instances with up to $n = 60$ variables are solved routinely by most algorithms. For the largest instances under consideration ($n = 100$), the success rates significantly deteriorate for the suites B and C. Note that suite A contains only 3 such large instances, which seem to be easier than the average instances from suites B and C.

The worst success rates are obtained for SAWEA. Better results are achieved by its enhanced variant LSAWEA, which consistently yields a higher SR and a lower AFS. However, both SAWEA and LSAWEA are inferior to the other algorithms. The algorithms FlipGA, ASAP, and WSAT yield comparable success rates but without clear dominance relations concerning AFS. The best SR on larger instances ($n \geq 75$) is obtained by RFEA2 and RFEA2+, with the only exception of ASAP on suite A.

The results show that evolutionary algorithms can compete with WSAT, and that some even exhibit higher success rates on all three benchmark suites.

Table 3: Results for benchmark suite A: SR and AFS.

Algorithm	$n = 30$		$n = 40$		$n = 50$		$n = 100$	
	SR	AFS	SR	AFS	SR	AFS	SR	AFS
SAWEA	1.00	34 015	0.93	53 289	0.85	60 743	0.72	86 631
RFEA2	1.00	3 535	1.00	3 231	1.00	8 506	0.99	26 501
RFEA2+	1.00	2 481	1.00	3 081	1.00	7 822	0.97	34 780
FlipGA	1.00	25 490	1.00	17 693	1.00	127 900	0.87	116 653
ASAP	1.00	9 550	1.00	8 760	1.00	68 483	1.00	52 276
WSAT	1.00	1 631	1.00	3 742	1.00	15 384	0.80	19 680

Table 4: Results for benchmark suite B: SR and AFS.

Algorithm	$n = 50$		$n = 75$		$n = 100$	
	SR	AFS	SR	AFS	SR	AFS
RFEA2	1.00	12 053	0.95	41 478	0.77	71 907
RFEA2+	1.00	11 350	0.96	39 396	0.81	80 282
FlipGA	1.00	103 800	0.82	29 818	0.57	20 675
ASAP	1.00	61 186	0.87	39 659	0.59	43 601
WSAT	0.95	16 603	0.84	33 722	0.60	23 853

Table 5: Results for benchmark suite C: SR and AFS.

Algorithm	$n = 20$		$n = 40$		$n = 60$		$n = 80$		$n = 100$	
	SR	AFS	SR	AFS	SR	AFS	SR	AFS	SR	AFS
SAWEA	1.00	12 634	0.89	35 988	0.73	47 131	0.52	62 859	0.51	69 657
LSAWEA	1.00	11 478	0.92	24 819	0.80	37 439	0.58	46 337	0.57	46 497
RFEA2	1.00	365	1.00	3 015	0.99	18 857	0.92	50 199	0.72	68 053
RFEA2+	1.00	365	1.00	2 951	0.99	19 957	0.95	49 312	0.79	74 459
FlipGA	1.00	1 073	1.00	14 320	1.00	127 520	0.73	29 957	0.62	20 319
ASAP	1.00	648	1.00	16 644	1.00	184 419	0.72	45 942	0.61	34 548
WSAT	1.00	334	1.00	5 472	0.94	20 999	0.72	30 168	0.63	21 331

5.2 Discussion

The experiments allow us to identify some features that contribute to the good performance of an EA for SAT. Among the EAs with adaptive fitness functions, the worst results are obtained for SAWEA, which is the only one that completely relies on a blind mutation operator. This indicates that adaptive fitness functions are not the single driving force in the success of such EAs, and that problem-specific variation operators are quite useful.

Nevertheless, adaptive fitness functions are very promising because classical GAs with the static MAXSAT fitness are clearly inferior to SAWEA. A recent study revealed that the constant weight growth in f_{SAW} makes the adaptation slow down in later stages of the search process, and that the introduction of decay factors helps to cope with this problem (Gottlieb and Voss, 2000). Despite using these decay factors and

RFEA's mutation operator inside SAWEA, the resulting EA was inferior to RFEA. This underlines that the design of appropriate adaptive fitness functions is crucial.

EAs with the simple MAXSAT fitness function exhibit a satisfactory performance when local search is used, in particular, together with adaptation mechanisms like in ASAP. This, again, confirms what has already been recognized in previous works on EAs for combinatorial optimization problems, namely that problem knowledge has to be used in order to make EAs competitive with local search algorithms.

The use of local search together with the simple MAXSAT fitness function and blind mutations yields a separation of exploration and exploitation of the search space. While local search is mainly responsible for exploitation, blind genetic operators are mainly responsible for exploration. Thus, knowledge is directly incorporated in the EA as local search module, which can be adapted or changed without modifying the genetic operators and fitness function.

Since local search forms the core of FlipGA and ASAP, these EAs achieve similar performance to the local search heuristic WSAT. Interestingly, WSAT is inferior to RFEA2 and RFEA2+, which use guided mutations but no strict local optimization. This indicates the usefulness of adaptive fitness functions that are explicitly designed to guide the search away from local optima, where other algorithms may get trapped.

The above discussion raises the question whether one should use a combination of local search with adaptive fitness functions. A preliminary experimental investigation of EAs for constraint satisfaction problems using both an adaptive fitness function (based on f_{SAW}) and local search indicates that this combination is not beneficial (Craenen et al., 2000).

Concerning general parameters used by the algorithms, it is striking that all use population size 1, except for RFEA and FlipGA that use 4 and 10, respectively. Thus, rather small values seem to work quite well. Further, only FlipGA makes use of crossover; this indicates that crossover is not dominant, and that local search steps are essential.

5.3 Further Results

Our experiments indicate that EAs are competitive with state-of-the-art heuristic algorithms for SAT like WSAT. Here we present additional results from literature that compare our selected EAs to other heuristics.

Bäck et al. (1998) experimentally compared SAWEA with a variant of GSAT (Selman et al., 1992) called WGSAT (Frank, 1996) on test suite A. WGSAT is based on the same basic idea as WSAT, but also uses clause weights to guide the search, as done in SAWEA. The results of the experiments indicate SAWEA outperforming WGSAT on the considered benchmark instances.

Rossi et al. (2000) tested ASAP on the DIMACS benchmark instances[4] that were also used by Resende and Feo (1996) to evaluate their GRASP (*greedy randomized search procedure*). GRASP is a general search technique that constructs a set of potential solutions by a randomized greedy heuristic and then improves them by a local search procedure. The results reported in Rossi et al. (2000) indicate that ASAP is competitive with GRASP concerning SR and running time, except for a few instances of the `AIM` and of the `PAR` classes.

Folino et al. (1998) proposed a rather different EA called CGW (*cellular genetic WSAT*), which employs a cellular genetic algorithm model (Whitley, 1993) and incorporates WSAT in the mutation operator. The individuals are organized as cells in a

[4] `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/cnf`

spatial lattice, and they are updated synchronously by crossover and a WSAT-like mutation. Hard randomly generated 3-SAT problems, in particular tests with $n = 256$ up to $n = 2048$ variables, have been considered in Folino et al. (1998) together with some problems of the DIMACS test suite. The authors report CGW to have a better convergence behavior than WSAT. Rossi et al. (2000) compared ASAP to CGW, indicating that both EAs could solve all problems, except for one where ASAP failed. In all successful runs, ASAP needed far fewer bit flips than CGW to obtain a solution.

6 Conclusion

Our results show that, despite the failure of classical genetic algorithms for SAT, evolutionary algorithms can nevertheless be quite successful if equipped with additional techniques; they can even compete with state-of-the-art heuristics like the popular WSAT local search algorithm.

A list of promising options that should be considered when solving SAT by an evolutionary algorithm is:

1. Bit string representation. This is the most natural representation that proved to be superior to all other representations suggested in literature.
2. Adaptive fitness functions. They guide the evolution by information learned during the previous search process. In particular, they help to focus on difficult clauses and to escape from local optima.
3. Knowledge-based genetic operators. They are goal-oriented and thus more effective than blind operators like classical standard bit mutation.
4. Local search. This helps to concentrate the search on promising solutions – the local optima.
5. Adaptive schemes for escaping local optima. As premature convergence at local optima is a serious risk when using local search or knowledge-based operators, local optima should be escaped (i) implicitly by adaptation of the fitness function or (ii) explicitly by adaptation of the neighborhood (e.g., by means of a tabu list).
6. Promising parameter choices. A good starting point for a new EA for SAT would be a small population size and the use of a mutation only scheme.

The above list presents a rich set of methodological guidelines; nevertheless there remain several interesting issues that could be pursued in future research work. As all reviewed EAs use random initialization, it would be interesting to check whether biasing the initial population by some other heuristic could result in a better overall performance. A candidate for such heuristic could be the randomized greedy phase of GRASP from Resende and Feo (1996), because its randomized construction mechanism permits generation of different candidate solutions.

Further, we need a better understanding of the properties that make some SAT instances difficult for EAs, while others are relatively easy. A characterization of different classes of SAT instances could also lead to the development of specialized EAs for these specific classes – like the approach of Warners (1999) that can solve the PAR family of the DIMACS instances in very short time. Finally, it would be challenging to design a multi-strategies framework consisting of specialized EAs based on different (local search) methods and/or genetic operators, together with a supervisor module responsible for sending the input SAT instance to selected EAs of the system that are "specialized" for that instance, and for coordinating those different EAs.

References

Bäck, T., Eiben, A., and Vink, M. (1998). A superior evolutionary algorithm for 3-SAT. In Saravanan, N., Waagen, D., and Eiben, A., editors, *Proceedings of the Seventh Annual Conference on Evolutionary Programming*. *Lecture Notes in Computer Science*, Volume 1477, pages 125–136, Springer, Berlin, Germany.

Battiti, R. and Protasi, M. (1998). Approximate algorithms and heuristics for MAX-SAT. In Du, D.-Z. and Pardalos, P., editors, *Handbook of Combinatorial Optimization*, pages 77–148, Kluwer, Boston, Massachusetts.

Brayton, R. et al. (1985). *Logic Minimization Algorithms for VLSI minimization*, Kluwer, Boston, Massachusetts.

Cook, S. (1971). The complexity of theorem-proving procedures. In *Proceedings of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, ACM, New York, New York.

Craenen, B. et al. (2000). Combining local search and fitness function adaptation in a GA for solving binary constraint satisfaction problems. In Whitley, D. et al., editors, *Proceedings of Genetic and Evolutionary Computation Conference*, page 381, Morgan Kaufmann, San Francisco, California.

De Jong, K. (1975). An analysis of the behaviour of a class of genetic adaptive systems. Doctoral Dissertation, University of Michigan, Ann Arbor, Michigan. Dissertation Abstract International 36(10), 5140B.

De Jong, K. and Spears, W. (1989). Using genetic algorithms to solve NP-complete problems. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 124–132, Morgan Kaufmann, San Mateo, California.

de Jong, M. and Kosters, W. (1998). Solving 3-SAT using adaptive sampling. In Poutré, H. L. and van den Herik, J., editors, *Proceedings of the Tenth Dutch/Belgian Artificial Intelligence Conference*, pages 221–228.

Dongarra, J. (1993). Performance of various computers using standard linear equations software. Technical Report CS89-85, Computer Science Department, University of Tennessee, Knoxville, Tennessee.

Du, D., Gu, J., and Pardalos, P., editors (1997). *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, AMS, Providence, Rhode Island.

Eiben, A. and van der Hauw, J. (1997). Solving 3-SAT with adaptive genetic algorithms. In *Proceedings of the Fourth IEEE Conference on Evolutionary Computation*, pages 81–86, IEEE Press, Piscataway, New Jersey.

Fleurent, C. and Ferland, J. (1996). Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In Trick, M. and Johnson, D. S., editors. *Second DIMACS Challenge, special issue*, Volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 619–652, AMS, Providence, Rhode Island.

Folino, G., Pizzuti, C., and Spezzano, G. (1998). Combining cellular genetic algorithms and local search for solving satisfiability problems. In *Proceedings of Tenth IEEE International Conference on Tools with Artificial Intelligence*, pages 192–198, IEEE, Piscataway, New Jersey.

Frank, J. (1994). A study of genetic algorithms to find approximate solutions to hard 3CNF problems. In Yfantis, A., editor, *Proceedings of Golden West International Conference on Artificial Intelligence*, Kluwer, Boston, Massachusetts.

Frank, J. (1996). Weighting for Godot: Learning heuristics for GSAT. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 338–343, AAAI Press, Menlo Park, California.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: a Guide to the Theory of NP-completeness*, Freeman, San Francisco, California.

Gottlieb, J. and Voss, N. (1998a). Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. In Eiben, A. et al., editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*. *Lecture Notes in Computer Science*, Volume 1498, pages 755–764, Springer, Berlin, Germany.

Gottlieb, J. and Voss, N. (1998b). Representations, fitness functions and genetic operators for the satisfiability problem. In Hao, J.-K. et al., editors, *Proceedings of Artificial Evolution*. *Lecture Notes in Computer Science*, Volume 1363, pages 55–68, Springer, Berlin, Germany.

Gottlieb, J. and Voss, N. (2000). Adaptive fitness functions for the satisfiability problem. In Schoenauer, M., editors, *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature*. *Lecture Notes in Computer Science*, Volume 1917, pages 621–630, Springer, Berlin, Germany.

Hao, J.-K. (1995). A clausal genetic representation and its evolutionary procedures for satisfiability problems. In Pearson, D., Steele, N., and Albrecht, R., editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 289–292, Springer, Vienna, Austria.

Hoos, H. (1998). *Stochastic Local Search – Methods, Models, Applications*. PhD thesis, Darmstadt University of Technology, Germany.

Larrabee, T. (1992). Efficient generation of test patterns using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1):4–15.

Larrabee, T. and Tsuji, Y. (1992). Evidence for a satisfiability threshold for random 3CNF formulas. Technical Report UCSC-CRL-92-42, University of California, Santa Cruz, California.

Marchiori, E. and Rossi, C. (1999). A flipping genetic algorithm for hard 3-SAT problems. In Banzhaf, W. et al., editors, *Proceedings of Genetic and Evolutionary Computation Conference*, pages 393–400, Morgan Kaufmann, San Francisco, California.

McAllester, D., Selman, B., and Kautz. H. (1997). Evidence for invariants in local search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 321–326, AAAI Press, Menlo Park, California.

Mitchell, D., Selman, B., and Levesque, H. (1992). Hard and easy distributions of SAT problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465, AAAI Press, Menlo Park, California.

Park, K. (1995). A comparative study of genetic search. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 512–519, Morgan Kaufmann, San Francisco, California.

Rana, S. and Whitley, D. (1998). Genetic algorithm behavior in the MAXSAT domain. In Eiben, A. et al., editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*. *Lecture Notes in Computer Science*, Volume 1498, pages 785–794, Springer, Berlin, Germany.

Resende, M. and Feo, T. (1996). A GRASP for satisfiability. In Trick, M. and Johnson, D. S., editors (1996). *Second DIMACS Challenge, special issue*, Volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 499–520, AMS, Providence, Rhode Island.

Rossi, C., Marchiori, E., and Kok, J. (2000). An adaptive evolutionary algorithm for the satisfiability problem. In Carroll, J. et al., editors, *Proceedings of ACM Symposium on Applied Computing*, pages 463–469, ACM, New York, New York.

Selman, B., Kautz, H., and Cohen, B. (1994). Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 337–343, AAAI Press, Menlo Park, California.

Selman, B., Levesque, H., and Mitchell. D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, AAAI Press, Menlo Park, California.

Singer, J., Gent, I., and Smaill, A. (2000). Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270.

Spears, W. (1996). Simulated annealing for hard satisfiability problems. In Trick, M. and Johnson, D. S., editors. *Second DIMACS Challenge, special issue*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 533–558, AMS, Providence, Rhode Island.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In Schaffer, J., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, Morgan Kaufmann, San Mateo, California.

Trick, M. and Johnson, D. S., editors (1996). *Second DIMACS Challenge, special issue*, Volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, AMS, Providence, Rhode Island.

Tseitin, G. (1968). On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, 2:115–125.

Warners, J. (1999). *Nonlinear Approaches to Satisfiability Problems*. PhD thesis, Eindhoven University of Technology, The Netherlands.

Whitley, D. (1993). Cellular genetic algorithms. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 658, Morgan Kaufmann, San Mateo, California.